一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

複数のプログラミング言語を対象とした テストの信頼不能性発生原因の調査

久保　光生†　肥後　芳樹†　楠本　真二†

† 大阪大学大学院情報科学研究科
E-mail: †{hkr-kubo,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　テストの信頼不能性とは，実行ごとに結果が変わる可能性のある性質を指す．このような性質を持つテストは現実のプロジェクトに広く存在し，開発者に混乱をもたらすことがある．テストの信頼不能性には様々な要因が寄与することが知られ，現在までに報告された要因は約 15 種類にのぼる．これまでの複数の研究により，プログラミング言語によって信頼不能テストの発生原因は異なることが示唆されている．これを確かめるための予備調査として，オープンソースの信頼不能テストのデータセットである iDoFT を用いて，Java および Python における信頼不能性の発生原因を調査した．調査の結果，発生原因の傾向が異なることが改めて確かめられ，その要因は Python と Java の言語仕様の違いであることが確かめられた．さらに，異なるプログラミング言語での調査のため，GitHub 上にある 1,000 個の Rust リポジトリを用いて，信頼不能テストに関するコミットの収集，および信頼不能テストの検出実験を行い，信頼不能性の発生原因を調査した．調査の結果，最も多い発生原因は非同期処理であることが判明した．また，以前の分類にあてはまらない新たな信頼不能性のカテゴリや，Rust の言語仕様により発生しなくなったと考えられる信頼不能性を発見した．

キーワード　信頼不能テスト, リポジトリマイニング

1. はじめに

回帰テストとは，コードの変更によって既存の機能に影響を及ぼさないことを確認するプロセスである．このテストは，既にテストされたプログラムが修正後も期待通りに動作するかを確認する．もしテストが失敗した場合，開発者はコードの変更に原因があると判断し，テスト対象のコード (CUT) に問題があるのか，テストコード自体を変更する必要があるのかを理解しようとする．開発者はコードの変更が既存の機能を破壊しないことを保証するために，テストが決定的な結果を持つことを前提にしている．

一方で，実行のたびに結果が変化する可能性のあるテストは，信頼不能テスト (Flaky Test) と呼ばれる．非決定的な結果を持つ信頼不能テストは，テストの失敗がコードの変更に起因するか否かを判断することを困難にし，開発者に混乱をもたらす．わずかな数の信頼不能テストであっても，その検出や修正には多大な作業が必要となる．例えば Google はテストの失敗のうち 4.56% が信頼不能テストに起因し，これらの信頼不能テストにより開発のプロセスが大幅に遅延していると報告している [1].

よって，信頼不能テストの性質を理解することは，それに対処するにあたってとても重要な課題である．これまでの複数の研究により，信頼不能テストの発生原因や修正方法が調査されており，約 15 種類のカテゴリに分類する方法が提案されている [1]〜[3]．また，プログラミング言語によって信頼不能テストの発生原因が異なることも示唆されている．これを確かめるため，本研究では予備調査としてオープンソースの信頼不能テストデータセットである iDoFT を用いて，Java および Python における信頼不能性の発生原因を s 分類した．分類の結果，プログラミング言語によって発生原因の傾向が異なることが改めて確かめられ，その要因は主に Python の言語仕様であることが確かめられた．

また，以前の研究では，Java，Python，JavaScript における信頼不能テストについて調査されている [1]〜[3] が，その他の言語における信頼不能テストについては調査されておらず，プログラミング言語による信頼不能テスト発生原因の差異について十分に調査されているとはいえない．

本研究では，近年盛んに使用されている Rust 言語における信頼不能テストの調査を行った．具体的には，991 個の Rust リポジトリにおける信頼不能テストに関するコミットの収集と，それらのリポジトリから無作為に抽出された 100 個のリポジトリにおける信頼不能テストの検出実験を行い，これらの方法で得られた信頼不能テストの分類を行った．

調査の結果，他の言語と同様に Rust にも信頼不能テストが存在することが判明した．また，発生件数が上位の原因は非同期処理，ネットワーク，順序依存であり，これらは他の言語でも多く見られる発生原因であることが判明した．

さらに，Rust に特有の新たな信頼不能テストのカテゴリや，Rust の言語仕様により発生しなくなったと考えられる信頼不能性を発見した．

2. 信頼不能テスト

信頼不能テスト (Flaky Test) とは，実行のたびに結果が変化する可能性のあるテストを指す．このような不安定さは，非同期処理やネットワークの状態など，非決定的な要素が原因で発生することが知られている．信頼不能テストは開発者の混乱を招き，開発のプロセスを大幅に遅延させる [1]．よって，その原因を解明し，安定したテストへと改善することが，効率的なソフトウェア開発に不可欠である．

2.1 信頼不能テストの分類

信頼不能テストにはいくつかの分類方法が存在する．

分類方法の 1 つに，順序依存 (Order-dependent) または非順序依存 (Non order-dependent) という 2 種類への分類がある [4]．順序依存とは，テストスイート内に複数のテストケースがある場合，テストケースの実行順序によって各テストケースの成否が変化する信頼不能テストを指す．非順序依存とは，単独で実行した場合でもテストケースの成否が変化する信頼不能テストを指す．

また，信頼不能性の発生原因に基づく分類も存在する．例えば Luo らは，彼らが調査した信頼不能性を分類するために，順序依存 (Test Order Dependency) を含む 10 種類のカテゴリを定義している [1]．また，Eck らは，Luo らが定義した 10 種類のカテゴリに加え，新たに 4 種類のカテゴリを，Hashemi らは新たに 4 種類のカテゴリを定義している [3], [5]．これらのカテゴリを表 1 に示す(注1)．

2.2 信頼不能テストの収集

信頼不能テストの収集には様々な方法が存在する．収集方法の 1 つに，Git リポジトリにおけるコミットの抽出がある．これは，リポジトリから特定のキーワードが含まれるコミットを抽出し，得られたコミットのメッセージや変更点などを目視で確認することで信頼不能テストを収集するという方法である．

また，テストの繰り返し実行も収集方法の 1 つである．テストを何度も実行し，1 回でも結果が変化したテストは信頼不能テストであると判断することができる．

他の方法として，繰り返し実行を伴わずに信頼不能テストを検出するツールの使用もある．例えば，iFixFlakies ではテストの実行順序を変更することで，DeFlaker では，コードの差分とカバレッジ情報を用いて信頼不能テストを検出する [6], [7].

2.3 各プログラミング言語における信頼不能テスト

各プログラミング言語における信頼不能テストを調査した先行研究が存在する．Luo らは 51 個のオープンソースプロジェクト（うち，44 個が Java に関係，残りの 7 個は C,C++,C#,Ruby,Python,Scala に関係）から抽出された 201 件のコミットを対象に，信頼不能テストの発生原因を調査した [1]．調査の結果，発生原因は 10 種類に分類でき，中でも非同期処理 (Async Wait)，同時実行性 (Concurrency)，テストの実行順序依存 (Test Order Dependency) が主な原因であることが示された．また，これらの発生原因に対処するための戦略についても考察を行った．

Gruber らは Python で書かれた 22,352 個のオープンソースプロジェクトを対象に信頼不能テストの検出実験を行った [2]．検出の結果，テストの実行順序依存が最も主要な発生原因であり，これは Luo らの調査とは異なることが示された．また，パッケージのインストールなど，コードの外部の要因で発生する信頼不能テストが一定数存在することが示された．

Hashemi らは Github 上でスター数上位の JavaScript プロ

(注1)：Hashemi らによって定義されたカテゴリである “OS”,“Platform” は Eck らによって定義された “Platform Dependency” のサブカテゴリとみなせるため，この表には含んでいない

表 1　これまでに提案された信頼不能性のカテゴリ

カテゴリ	出典	説明	プログラミング言語		
			Java	Python	JS
Async Wait	[1]	非同期処理の待機時間の差によって結果が変化する．	○	○	○
Concurrency	[1]	並列処理の同時実行によって結果が変化する．	○	○	○
Test Order Dependency	[1]	テストスイートに含まれる各テストケースの実行順序により結果が変化する．	○	○	○
Resource Leak	[1]	リソースへの同時アクセスやクローズ忘れなどによって結果が変化する．	○	○	○
Network	[1]	ネットワークの状態やネットワーク上のリソースの変化により結果が変化する．	○	○	○
Time	[1]	現在時刻によって結果が変化する．	○	○	○
IO	[1]	入出力の状態によって結果が変化する．	○	○	○
Randomness	[1]	テスト対象のコードが持つランダム性によって結果が変化する．	○	○	○
Floating Point Operations	[1]	浮動小数点数の誤差によって結果が変化する．	○	○	○
Unordered Collections	[1]	コレクションに含まれる要素の順序が変化することにより結果が変化する．	○	△*	○
Too Restrictive Range	[5]	許容される値の範囲が狭すぎることにより結果が変化する．			
Test Case Timeout	[5]	テストケースのタイムアウトの有無により結果が変化する．		○	
Platform Dependency	[5]	OS 自体の違いや OS のバージョンの差により結果が変化する．			○
Test Suite Timeout	[5]	テストスイートのタイムアウトの有無により結果が変化する．			
HardWare	[3]	ハードウェアの違いにより結果が変化する．			○
UI	[3]	ユーザーインターフェースに関連して結果が変化する．			○

* Python3.3 から Python3.5 のみで発生

ジェクトから抽出された 452 件のコミットを対象に信頼不能テストを調査した [2]．調査の結果，発生原因は 13 種類に分類でき，中でも非同期処理 (Async Wait)，同時実行性 (Concurrency)，OS の違いが主要な要因であることが示された．また，プラットフォーム，UI，ハードウェアといったカテゴリが新たに定義された．

これらの先行研究の調査結果により，プログラミング言語によって信頼不能テストの発生原因は異なることが示唆される．

3. 予備調査

予備調査では，信頼不能テストのデータセットである iDoFT(注2) に含まれるテストおよび修正コミットを目視し，各テストの分類を行った．

3.1 調査方法

信頼不能テストデータセットである iDoFT の 2024 年 4 月 8 日における版に含まれるテストのうち，修正コミットが存在するテストの分類を行った．

iDoFT には，Java 及び Python の信頼不能テストが記録されている．これらのテストのうち多くは信頼不能テスト検出ツールである iDFlakies [8] や NonDex [9] を用いて検出された．iDoFT に含まれる信頼不能テストのテストケースの一部には，修正を提案するプルリクエストのリンクが付随する．これにより，プルリクエストのリンクが存在する，すなわち修正コミットが存在する信頼不能テストのみを対象とすることができる．修正コミットが存在する信頼不能テストは，Java では 1,623 個，Python では 114 個存在した．

修正コミットが存在する信頼不能テストについて，プルリクエストのメッセージや修正の内容をもとに，各テストを表 1 に示すカテゴリに分類した．また，どのように分類してよいかわからない場合は「不明」とした．

3.2 調査結果

Java における分類の結果を表 2 に，Python における分類の結果を表 3 に示す．Java では約 76.2% のテストケースが Unordered Collection 順序付けされていないコレクション）に分類され，約 21.1% のテストケースが Test Order Dependency（テストの実行順序依存）に分類された．一方で，Python では約 95.6% のテストケースが Test Order Dependency（テストの実行順序依存）に分類された．

表 2　Java における信頼不能テストの分類結果

カテゴリ	テストケース数
Unordered Collections	1238
Test Order Dependency	344
Network	29
Time	7
Async Wait	3
Floating Point Operations	1

(注2)：http://mir.cs.illinois.edu/flakytests

3.3 考察

3.3.1 Java・Python 間の分類結果の差

iDoFT の中には，Unordered Collection（順序付けされていないコレクション）にあてはまる Python の信頼不能テストは存在しなかった．これは，Java の `HashSet` や `HashMap` のような順序を保証しないコレクションに対応する python のコレクション (`set` や `dict`) は，python3.6 以降においては順序を保証することが要因であると考えられる．

3.3.2 先行研究の調査結果との差

主に Java のプロジェクトを用いて行われた Luo らによる調査によると，最も多い発生原因は Async Wait（非同期処理）である．この結果は，予備調査の Java での分類結果とは異なる．これは，Luo らによる調査手法はコミットメッセージをもとにした目視確認であったのに対し，iDoFT には，主に既存の検出ツールを用いて発見された信頼不能テストが記録されていることが要因であると考えられる．

一方で，Python のプロジェクトを用いて行われた Gruber らによる調査において，最も多い発生原因は Test Order Dependency（テストの実行順序依存）であり，予備調査の Python での分類結果と一致する．これは，Gruber らによる調査では，テストの順序を変えて実行するなど，既存の検出ツールと似たアプローチを用いて信頼不能テストを検出していることが要因であると考えられる．

4. 調査

Rust は 2010 年に登場した比較的新しいプログラミング言語である [10]．Rust には所有権と呼ばれる概念が存在し，これを用いてメモリ操作の安全性をコンパイル時に検査することができる．また，Rust は C 言語や C++ に代わる言語を目指して開発されたため，Linux カーネルの開発などの低レイヤのプロジェクトにも用いられている．

Rust 言語を対象とした信頼不能テストについて調査するため，GitHub 上の Rust リポジトリから信頼不能テストに関するコミットの抽出を行った．また，開発者により発見されていない信頼不能テストを発見するため，信頼不能テストの検出実験を行った．その後，2 つの方法によって得られた信頼不能テストの分類を行った．調査の流れを図 1 に示す．

4.1 調査対象

調査対象は GitHub 上に存在する Rust リポジトリのうち，スター数が上位 1,000 個のリポジトリである．リポジトリ中には Rust のソースコードは存在するが，ビルドシステムである

表 3　Python における信頼不能テストの分類結果

カテゴリ	テストケース数
Test Order Dependency	109
Async Wait	2
Randomness	1
Too Restrictive Range	1
不明	1

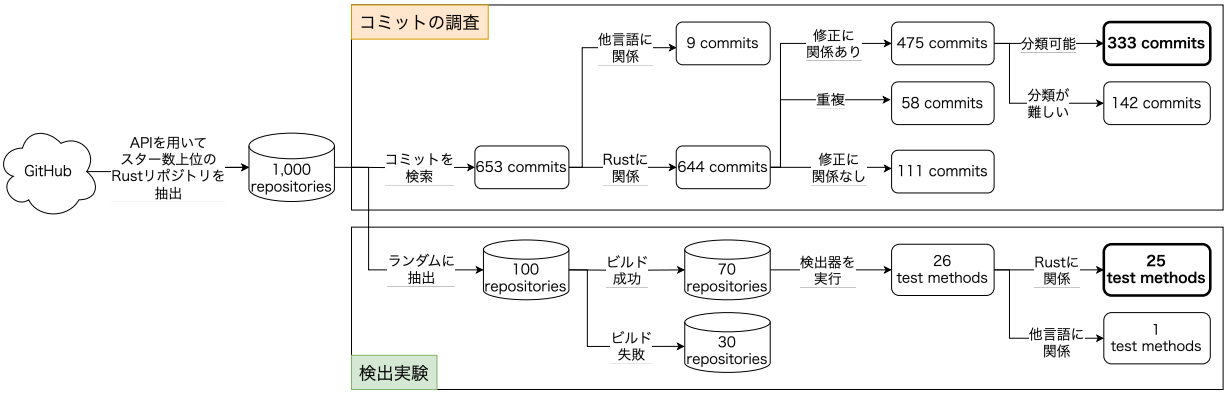

図 1　調査の流れ

cargo に対応しておらず，プロジェクトとして認識されない場合がある．これらを除外した結果，合計で 991 個のリポジトリが調査対象となった．

4.2　信頼不能テストに関するコミットの抽出

合計で 991 個のリポジトリから “flaky test” というキーワードを含むコミットを抽出した．コミットで変更されたテストコードやコミットメッセージをもとに，そのコミットが信頼不能テストの修正に関係するか，また他のコミットと重複していないかを確認した．

4.3　信頼不能テストの検出実験

検出実験に際し，Rust 用の信頼不能テスト検出器を作成した．このツールは Rust のパッケージ管理システムである cargo の機能を用いて，プロジェクト内にあるすべてのテストを実行する．これを何度も繰り返し実行してテストの成否を記録することで，1 回でも結果が変化したテストを信頼不能テストとして検出する．なお，cargo の機能でテストを実行する場合，すべてのテストが並列に実行されるため，実行の順序を制御することはできない．

Rust はコンパイルに長い時間を要する言語であり，テストの繰り返し実行にも多くの時間を要する．よって，991 個のプロジェクトから 100 個のプロジェクトを無作為に抽出し，デフォルトブランチの最新のコミットに対して信頼不能テスト検出器を実行した．実行には約 119 時間を要した．なお，検出対象のテストはユニットテストのみとし，ドキュメンテーションテストについては実行しない．検出器はテストを 100 回実行し，1 回でも成否情報が変化したテストを信頼不能テストとして記録する．

4.4　信頼不能テストの分類

検出器の実行により検出された信頼不能テスト，および信頼不能テストに関するコミットを目視確認し，分類を行った．分類に際しては，コード以外に，検出された信頼不能テストについてはテスト失敗時のエラーメッセージを，コミットから収集された信頼不能テストについてはコミットメッセージやプルリクエストのコメントを参考にした．テストの分類には，予備調査と同様に表 1 に示すカテゴリを用いた．また，これらのカテゴリに当てはまらない場合には，新たなカテゴリを定義した．

5.　調 査 結 果

5.1　信頼不能テストに関するコミットの確認結果

991 個のリポジトリのうち，信頼不能テストに関するコミットが存在するリポジトリは 128 個であり，これらのリポジトリから合計で 653 件のコミットが検出された．これらを目視で確認し，重複したコミットを除くと，475 件が信頼不能テストの修正や削除に関するコミットであることが判明した．このうち，信頼不能性の発生原因が特定できたコミットは 333 件であった．

5.2　信頼不能テスト検出実験の結果

100 個のリポジトリのうち，30 個でビルドエラーが生じた．ビルドが成功し，信頼不能テスト検出器を適用できたリポジトリは 70 個であった．これらの 70 個のリポジトリのうち，10 個のリポジトリから合計で 25 個の信頼不能テストが検出された．このうち 1 個の信頼不能テストは Rust ではなく別言語のコードが原因で発生したため，分類の対象から除外した．なお，コミットから抽出された信頼不能テストと重複する信頼不能テストは検出されなかった．

5.3　信頼不能テストの分類結果

コミットの抽出および検出実験で発見された信頼不能テストの分類結果を表 4 に示す．

出現数が最も多い発生原因は，検出器の実行により検出された信頼不能テストでは Test Order Dependency（テストの実行順序依存），コミットから収集された信頼不能テストでは Async Wait（非同期処理）であった．

検出器の実行により検出された信頼不能テストは，すべてが表 1 のカテゴリに当てはまった．信頼不能テストに関するコミットからは，新たな分類である “System” に当てはまる信頼不能テストが見つかった．

5.4　新たな分類 (System)

新たな分類である System は，システムに関する操作を行った際に，実行するたびに変化する値をアサーションに用いることにより発生する信頼不能性である．System に当てはまる信頼不能性の例として，動的メモリ確保を行う Rust コードを図 2 に示す．図 2 の 4 行目では，`alloc_id.0.get()` で確保しようとするメモリのアドレスを取得している．メモリのアド

```
if self.ev.ecx.check_mplace(mplace).is_err() {
    let (alloc_id, _, _) = self.ev.ecx.ptr_get_alloc_id(mplace.ptr)?;
    return Ok(MValue::Unallocated {
      alloc_id: Some(alloc_id.0.get()),
    });
}
```

図 2　動的メモリ確保を行う Rust コード

レスはシステムによって決定されるため，この値は実行のたびに変化する．よって，この値を用いたアサーションを行うテストは信頼不能テストとなる．

5.5　その他の分類に当てはまる信頼不能テスト

検出器の実行により検出された信頼不能テストおよびコミットから抽出された信頼不能テストの両者において，新たな分類 (System) に当てはまらなかった信頼不能テストは，すべて表 1 内のカテゴリに当てはまった．また，これらのカテゴリの中には Rust に特有の信頼不能性は見つからなかった．

一方で，Test Order Dependency にあてはまる信頼不能テストを目視することで，他の言語では見られるが Rust では見られない信頼不能性があることが判明した．Test Order Dependency には更に細かい発生原因が複数存在し，その 1 つにグローバル変数による信頼不能性がある．例として，予備調査で見つかった，Python においてグローバル変数のリセットを忘れることにより発生する信頼不能テストを図 3 に示す．このコードでは，グローバル変数として `df` を定義しており，それを複数のテストメソッドで使用している．あるテストメソッド中で `df` の値が変化すると，他のテストメソッド内で用いる `df` が定義時と異なる値になるため，テストの実行順序によってテストケースの成否が変化する．このような信頼不能性は予備調査において，Java および Python での存在が確認

表 4　コミットの抽出および検出実験で発見された信頼不能テストの分類結果

カテゴリ	コミットの抽出 *1	検出実験 *2
Async Wait	140	3
Network	70	6
Test Order Dependency	40	10
Platform Dependency	26	
Concurrency	20	
Unordered Collections	10	
Resource Leak	6	1
Randomness	6	1
IO	4	
Time	4	
UI	3	2
Floating Point Operations	2	1
System	2	

*1 単位はコミット数
*2 単位はテストメソッド数

```
df = read_dataset("mediation")
...
#Flaky Test
def test_logistic_regression(self):
    lom = logistic_regression(df["X"], df["Ybin"], as_dataframe=False)
    assert_equal(np.round(lom["coef"], 4), [1.3191, -0.1995])
    ...

#Flaky Test
def test_mediation_analysis(self):
    ma = mediation_analysis(data=df, x="X", m="M", y="Y", n_boot=500)
    ...
    assert_equal(ma["coef"].round(4).to_numpy(), [0.5610, 0.6542, 0.3961, 0.0396, 0.3565])
    ...
```

図 3　Python においてグローバル変数のリセットを忘れることにより発生する信頼不能テストのコード

されたが，本調査では見つからなかった．

6. 考　　察

6.1　分類結果

非同期処理，ネットワークなどどによる信頼不能性は，他のプログラミング言語における先行研究と同様に本研究でも多く見つかり，これらのカテゴリの中には Rust に特有の信頼不能性は見つからなかった．このことから，このような信頼不能性はプログラミング言語の仕様に関係なく発生すると考えられる．

グローバル変数に起因する信頼不能性は本調査では発見されなかった．これは Rust における変数の仕様が他の言語と異なることに要因があると考えられる．Python などの言語では，図 3 の 1 行目のように，可変なグローバル変数を定義することができる．一方で Rust では，可変なグローバル変数を定義しようとすると，コンパイルが失敗する．このように，Rust 言語における変数の仕様に関する制約により，グローバル変数に起因する信頼不能性は見られなかったと考えられる．

6.2　新たな分類が検出された理由

Java はプラットフォームに依存しない JVM 上で動作する言語であり，Python や JavaScript はインタプリタ型言語であるため，これらの言語でシステムの低レイヤにアクセスするコードが書かれることは非常に少ない．一方で，Rust ではポインタへのアクセスなど，システムの低レイヤにアクセスするコードが書かれることがある．このことから，システムの低レイヤにアクセスする際に起こる信頼不能性が新たに検出されたと考えられる．

6.3　検出実験で新たな発生原因が見つからなかった理由

検出実験で得られた信頼不能テストからは，信頼不能性の新たな発生原因は見つからなかった．これは，本調査では 991

個のリポジトリから 100 個のみを抽出して検出実験を行ったため，サンプル数が少ないことが要因の 1 つとして考えられる．また，Alshammari らの研究により，すべての信頼不能テストが 100 回程度の繰り返し実行のみで得られるのではなく，繰り返し実行のみでは再現が難しい信頼不能テストが存在することが示されている [11]．このことから，繰り返し実行のみによる検出実験では信頼不能テストを十分に発見できないことも要因の 1 つとして考えられる．

7. 妥当性への脅威

7.1 内的妥当性

本研究では信頼不能テストに関するコミットを抽出するために “flaky test” というキーワードを用いた．しかし，Hashemi らによる研究では，信頼不能テストに関するキーワードとして他に “flaky”,“flakiness”,“intermit” などが挙げられている [3]．本研究ではこれらのキーワードに当てはまるコミットからは信頼不能テストを抽出できていない．

本研究で使用した信頼不能テスト検出器は，並列にテストを繰り返し実行することで信頼不能テストを検出している．よって，テストを複数回実行するだけでは検出できない信頼不能テストや，順序依存の信頼不能テストを検出できていない可能性がある．また，繰り返しの回数が少ないことから，信頼不能テストが十分に検出できていない可能性がある．

7.2 外的妥当性

本研究で使用した Rust プロジェクトは GitHub におけるスター数上位 1,000 個のプロジェクトである．調査対象のプロジェクトを変更すると，本研究における実験結果とは異なる結果が生じる可能性がある．

8. おわりに

本研究では，信頼不能テストの発生原因がプログラミング言語ごとに異なることを確かめるため，予備調査として iDoFT に含まれる信頼不能テストの分類を行った．分類の結果，発生原因はプログラミング言語ごとに異なることが改めて確かめられた．

また，本研究では他のプログラミング言語での調査のため，Rust のリポジトリを用いてコミットの抽出および信頼不能テストの検出実験を行った．調査の結果，リポジトリから 333 件の分類可能な信頼不能テストに関連するコミットが発見され，検出実験では 25 個の信頼不能テストメソッドが発見された．これらの発生原因を分類することにより，Rust における発生件数が上位の原因は非同期処理，ネットワーク，順序依存であり，これらは他のプログラミング言語でも多く見られる発生原因であることが判明した．また，Rust に特有の新たな信頼不能テストのカテゴリや，Rust の言語仕様により発生しなくなったと考えられる信頼不能性を発見した．

今後の展望として，信頼不能テスト検出器を本研究で収集した 991 個のリポジトリ全てにおいて実行し，Rust に特有の信頼不能性を発見することが挙げられる．また，それに伴い，信頼不能テスト検出器の機能拡張も予定している．現在の検出器にはテストを繰り返し実行する機能しかないが，実行するテストの順序を任意に変える機能を追加することにより，順序依存の信頼不能テストを発見できると考えられる．また，ネットワークの状態や入出力の状態を切り替える機能を追加することにより，単純な繰り返し実行では再現が難しい信頼不能テストを検出できると考えられる．

加えて，信頼不能テストの自動修正方法の提案も今後の展望の 1 つである．具体的には，iDoFT に含まれる信頼不能テストや，本研究で見つかった信頼不能テストの修正パターンを用いた大規模言語モデルのファインチューニングなどを検討している．

謝辞 本研究は JSPS 科研費（JP24H00692，JP21K18302，JP21H04877，JP23K24823，JP22K11985）の助成を得て行われた．

文　献

[1] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, “An empirical analysis of flaky tests,” 22nd ACM SIGSOFT international symposium on foundations of software engineering, pp.643–653, 2014.

[2] M. Gruber, S. Lukasczyk, F. Kroiß, and G. Fraser, “An empirical study of flaky tests in python,” 14th IEEE Conference on Software Testing, Verification and ValidationIEEE, pp.148–158 2021.

[3] N. Hashemi, A. Tahir, and S. Rasheed, “An empirical study of flaky tests in javascript,” IEEE International Conference on Software Maintenance and EvolutionIEEE, pp.24–34 2022.

[4] S. Zhang, D. Jalali, J. Wuttke, K. Muşlu, W. Lam, M.D. Ernst, and D. Notkin, “Empirically revisiting the test independence assumption,” International Symposium on Software Testing and Analysis, pp.385–396, 2014.

[5] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli, “Understanding flaky tests: the developer’s perspective,” 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.830–840, 2019.

[6] A. Shi, W. Lam, R. Oei, T. Xie, and D. Marinov, “ifixflakies: A framework for automatically fixing order-dependent flaky tests,” 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp.545–555, 2019.

[7] J. Bell, O. Legunsen, M. Hilton, L. Eloussi, T. Yung, and D. Marinov, “Deflaker: Automatically detecting flaky tests,” 40th international conference on software engineering, pp.433–444, 2018.

[8] W. Lam, R. Oei, A. Shi, D. Marinov, and T. Xie, “idflakies: A framework for detecting and partially classifying flaky tests,” 12th ieee conference on software testing, validation and verificationIEEE, pp.312–322 2019.

[9] A. Gyori, B. Lambeth, A. Shi, O. Legunsen, and D. Marinov, “Nondex: A tool for detecting and debugging wrong assumptions on java api specifications,” 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp.993–997, 2016.

[10] N.D. Matsakis and F.S. Klock, “The rust language,” ACM SIGAda Ada Letters, vol.34, no.3, pp.103–104, 2014.

[11] A. Alshammari, C. Morris, M. Hilton, and J. Bell, “Flakeflagger: Predicting flakiness without rerunning tests,” IEEE/ACM 43rd International Conference on Software Engineering, pp.1572–1584, 2021.