Fine-Grained Test Minimization

Arash Vahabzadeh, Andrea Stocco, Ali Mesbah

ICSE ’18, May 27-June 3, 2018, Gothenburg, Sweden

発表者　吉富　楓雅

目次

- テストの冗長性
- アプローチとRQ
- RQの結果
- 結論とまとめ

要約

ソフトウェアシステム開発が進むと，テストコードの冗長性が促進される．現在の技術では，コードカバレッジを利用して，テストスイートからテストケース全体を削除する方法をとっているが，この方法では
(1)アサーションを同時に削除することでテストの有効性が阻害される．
(2)部分的に冗長なテストケースが無視される．
そこで本論文ではテストケース最小化のための新しいアプローチを提案する．

テストの冗長性

背景

- ソフトウェアシステムを開発する際，品質と正確性を保証するために**テスト**が行われる

  * プロダクトコードが進化すると，新しいテストケースが追加され，既存のテストコードは変更される

テストの冗長性

- テストコードの開発が進むと「**冗長化**」が発生する

  ＊ **冗長化：似たようなコードが繰り返し記述されること**

- 冗長化が進むと，保守や回帰テストを行う際にコストが余計にかかる

- **コストを削減するには，冗長化を取り除く必要がある**

```
1   @Test
2   public void testAdd() {
3    Complex x = new Complex(3.0, 4.0);
4    Complex y = new Complex(5.0, 6.0);
5    Complex z = x.add(y);
6    assertEquals(8.0, z.getReal(), 1.0e-5);
7   }
8   @Test
9   public void testSubtract() {
10   Complex x = new Complex(3.0, 4.0);
11   Complex y = new Complex(5.0, 6.0);
12   Complex z = x.subtract(y);
13   assertEquals(-2.0, z.getReal(), 1.0e-5);
14  }
15  @Test
16  public void testMultiply() {
17   Complex x = new Complex(3.0, 4.0);
18   Complex y = new Complex(5.0, 6.0);
19   Complex z = x.multiply(y);
20   assertEquals(-9.0, z.getReal(), 1.0e-5);
21  }
22  @Test
23  public void testDivide() {
24   Complex dividend = new Complex(3.0, 4.0);
25   Complex divisor = new Complex(5.0, 6.0);
26   Complex q = dividend.divide(divisor);
27   assertEquals(39.0 / 61.0, q.getReal(), 1.0e-5);
28  }
```

図１：冗長なステートメントを持つテストケース

既存手法と問題点

- 既存手法：**コードカバレッジ**などの妥当性基準を，冗長なカバレッジでテストケース全体を削除するためのガイドラインとして使用[1]

- 既存手法の問題点
  - **アサーションを同時に削除してしまうことでテストの有効性が阻害される**
  - **部分的に冗長なテストケースが無視される**

[1]Shin Yoo and Mark Harman. 2012. Regression Testing Minimization, Selection and Prioritization: A Survey. Software Testing, Verification and Reliability 22, 2 (March 2012), 67–120. http://dx.doi.org/10.1002/stv.430

問題の克服

- テストステートメント内のより詳細な冗長性をターゲットとするアプローチが必要
  - テストケースをリファクタリングするのは困難
  - 複雑なテストケースを手動で並び替えるのは容易ではない
- 本論文では，**ステートメントレベルでテストケースを分析**し，テストステートメント間の関係をキャプチャするテストモデルを構築
  - 重複したテストステートメントの特定や削除など，細かいテストの作業が可能

使用ツール

- Testler[41](https://github.com/saltlab/Testler)

  * SALT labが作成したツール

    - Javaテストの細かい冗長性を排除するテストスイートモデルを作成

    - JUnit4をサポート

冗長性の除去

```java
@Test
public void testAdd() {
 Complex x = new Complex(3.0, 4.0);
 Complex y = new Complex(5.0, 6.0);
 Complex z = x.add(y);
 assertEquals(8.0, z.getReal(), 1.0e-5);
}
@Test
public void testSubtract() {
 Complex x = new Complex(3.0, 4.0);
 Complex y = new Complex(5.0, 6.0);
 Complex z = x.subtract(y);
 assertEquals(-2.0, z.getReal(), 1.0e-5);
}
@Test
public void testMultiply() {
 Complex x = new Complex(3.0, 4.0);
 Complex y = new Complex(5.0, 6.0);
 Complex z = x.multiply(y);
 assertEquals(-9.0, z.getReal(), 1.0e-5);
}
@Test
public void testDivide() {
 Complex dividend = new Complex(3.0, 4.0);
 Complex divisor = new Complex(5.0, 6.0);
 Complex q = dividend.divide(divisor);
 assertEquals(39.0 / 61.0, q.getReal(), 1.0e-5);
}
```

例：Apache Commons MathプロジェクトのComplexTestクラス

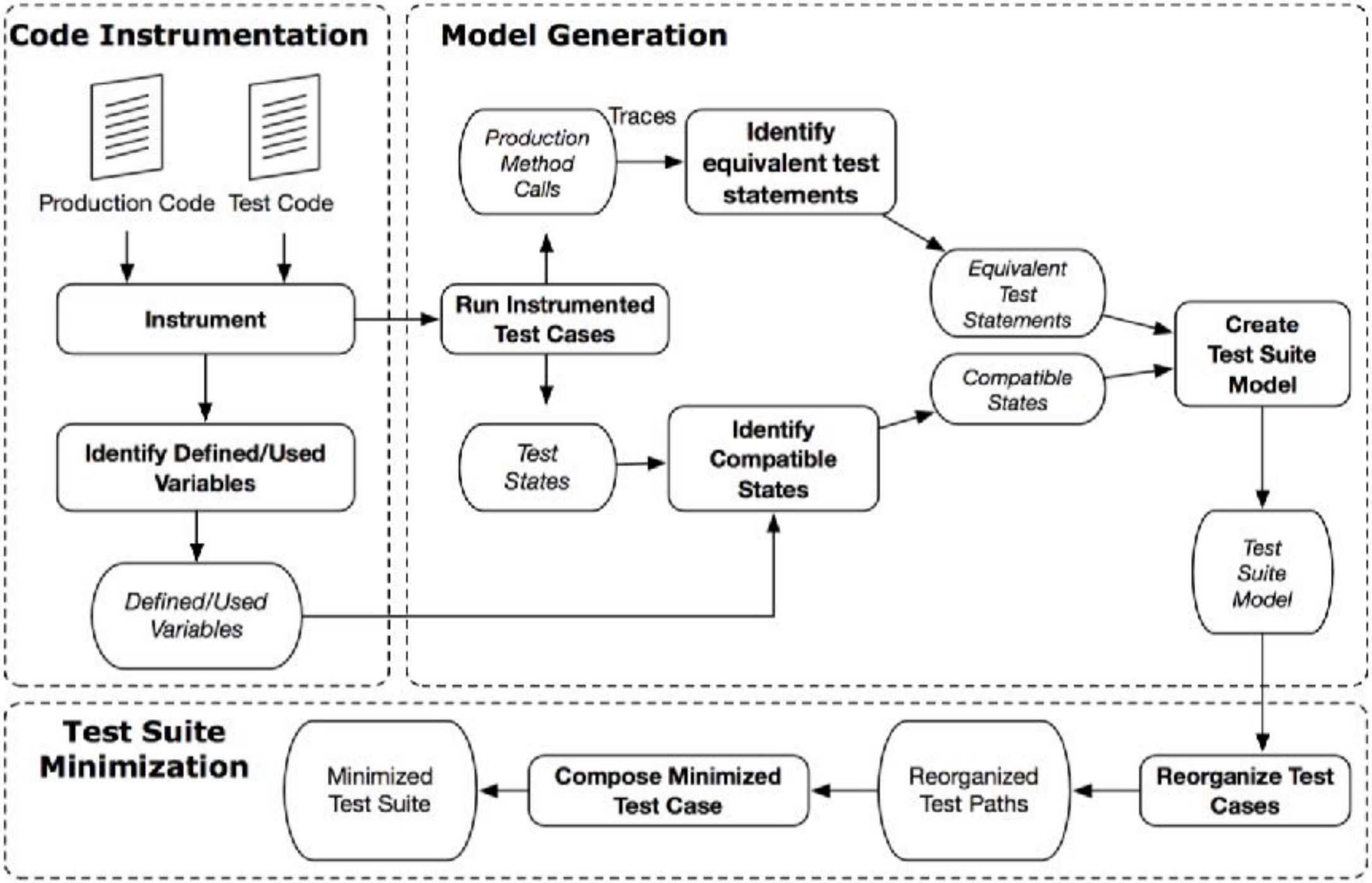
* 冗長なステートメントを持つ4つのテストケースで構成（xとyの初期化）

```java
@Test
public void testAdd_Subtract_Multiply_Divide() {
 Complex x = new Complex(3.0, 4.0);
 Complex y = new Complex(5.0, 6.0);
 Complex z = x.add(y);
 assertEquals(8.0, z.getReal(), 1.0e-5);
 Complex z_subtract = x.subtract(y);
 assertEquals(-2.0, z_subtract.getReal(), 1.0e-5);
 Complex z_multiply = x.multiply(y);
 assertEquals(-9.0, z_multiply.getReal(), 1.0e-5);
 Complex q = x.divide(y);
 assertEquals(39.0 / 61.0, q.getReal(), 1.0e-5);
}
```

図２：冗長性の除去

アプローチとRQ

研究のアプローチ


Code Instrumentation
Production Code
Test Code
Instrument
Identify Defined/Used Variables
Defined/Used Variables
Model Generation
Production Method Calls
Traces
Identify equivalent test statements
Equivalent Test Statements
Run Instrumented Test Cases
Test States
Identify Compatible States
Compatible States
Create Test Suite Model
Test Suite Model
Test Suite Minimization
Reorganize Test Cases
Reorganized Test Paths
Compose Minimized Test Case
Minimized Test Suite

各工程について

- **コード計測(Code Instrumentation：CI)**

1. 参照可能なすべての変数の型と値をコード計測機に格納

   - プロダクションメソッド呼び出し(PMC)を拾うために，入力値および各メソッドの終了ポイントを記録するためのプロダクトコードを計測

   - 各テストステートメントによって呼び出されるプロダクションメソッドを追跡するために，すべてのテストケースのステートメントごとに別々の呼び出しスタックログを生成

2. 定義あるいは使用されている，参照可能なすべての変数を特定する

   - オブジェクトの静的変数やメンバ変数など

各工程について

- **モデル生成(Model Generation：MG)**
  - 計測の後，プロダクションコードに対してテストを実行
    - トレースを使用してテストステートメントのセットを計算
- テストが依存状態なのかを確認

各工程について

- **テストスイートの最小化(Test Suite Minimization：)**

  * 推論モデルを使用して，重複したテストステートメントを識別し削除する

    - 図１を例に：4つのテストケースを再構成することで，図２のような最小限のテストケースを作成

    - このテストケースは同じカバレッジとアサーションを持つが６つステートメントが少ない

リサーチクエスチョン

- RQ1：部分的に冗長なテストはどの程度普及しているか？
- RQ2：Testlerによってどの程度冗長性が削除されるのか？
- RQ3：Testlerによって再編成されたテストスイートの実行時間はどのくらい短縮しているか？
- RQ4：Testlerはテストスイートのコードカバレッジと障害検出を保持することができるのか？
- RQ5：Testlerにより各工程にはどの程度時間を要するのか？

RQの結果とまとめ

RQ1：部分的に冗長なテストはどの程度普及しているか？

	Lines of Code (K)		Test Cases		
	Prod.	**Test**	**#**	**Par. Redundant**	**%**
Collections	12.3	20.3	459	110	24
Math	45.2	59.1	3,990	1,354	34
Lang	26.6	41.6	2,344	604	26
Email	2.7	3.0	141	21	15
AssertJ	6.4	24.7	4,620	947	20
CheckStyle	16.6	27.0	1,865	164	9
PMD	42.2	14.3	465	192	41
LambdaJ	3.6	4.8	260	82	32
Java-library	24.3	12.2	611	86	14
JFreeChart	98.5	17.3	187	111	59
Accumulo	192.9	13.8	987	289	29
XmlSecurity	40.0	32.3	941	190	20
Crunch	24.0	7.2	452	78	17
Tika	60.5	17.1	553	75	14
Log4J	46.2	25.0	1,475	336	23
Total/Average	642.0	319.6	19,350	4,639	24

RQ2：Testlerによってどの程度冗長性が削減されるのか？

	Redundancy						Execution Time			Code Coverage				Fault Detection		Analysis Time
	Par. Red. Tests			Red. Test Stmts			Seconds		%	Stmt		Branch		Mut. Score		CI/MG/TR (Tot)
	Before (#)	Reduced (#)	Reduction (%)	Before (#)	Reduced (#)	Reduction (%)	Original	Reorganized	Reduction	Before (%)	Reduced (%)	Before (%)	Reduced (%)	Before (%)	Reduced (%)	Time (seconds)
Collections	110	52	47	210	165	79	17.359	16.440	5	84.7	84.7	72.3	72.3	40.0	40.0	71/70/183 (324)
Math	1,354	946	70	4,052	1,623	40	83.109	72.804	12	92.7	92.7	85.7	85.7	79.0	79.0	101/401/1103 (1,605)
Lang	604	335	55	1,533	563	37	11.053	10.376	6	92.9	92.9	57.5	57.5	66.0	66.0	38/43/372 (453)
Email	21	10	48	38	12	32	7.143	5.933	17	54.6	54.6	50.9	50.9	48.0	48.0	10/12/3 (25)
AssertJ	947	253	27	516	207	40	3.389	2.794	18	95.6	95.6	92.2	92.2	63.0	63.0	244/37/383 (664)
CheckStyle	164	77	47	258	122	47	26.361	21.035	20	95.4	95.4	96.8	96.8	17.0	17.0	172/625/216 (1,013)
PMD	192	122	64	346	203	59	7.550	7.145	5	90.1	90.1	52.8	52.8	25.0	25.0	367/110/32 (508)
LambdaJ	82	42	51	109	55	50	1.468	0.924	37	85.2	85.2	52.2	52.2	85.0	85.0	12/6/13 (31)
Java-library	86	37	43	338	71	21	2.062	1.851	10	96.0	96.0	60.7	60.7	54.0	54.0	58/97/215 (370)
JFreeChart	111	56	50	1,657	374	23	2.752	2.485	10	95.5	95.5	47.6	47.6	24.0	24.0	45/52/205 (302)
Accumulo	289	93	32	683	140	20	65.883	50.794	23	94.7	94.7	71.5	71.5	20.0	20.0	58/333/106 (497)
XmlSecurity	190	156	82	1,102	1,079	98	68.590	57.590	16	78.7	78.7	54.0	54.0	68.0	68.0	42/179/142 (363)
Crunch	78	52	67	125	83	66	161.812	137.657	15	94.5	94.5	74.4	74.4	31.0	31.0	27/123/20 (170)
Tika	75	45	60	141	54	38	78.970	54.350	31	88.4	88.4	57.5	57.5	35.0	35.0	50/673/348 (1,071)
Log4J	336	127	38	711	290	41	261.987	198.343	24	80.5	80.5	38.7	38.7	13.0	13.0	48/490/100 (638)
Total/Average	4,639	2,403	52	11,819	5,041	43	799.488	640.521	20	88.0	88.0	64.3	64.3	44.5	44.5	8,034/536

Testlerの実行によって，平均52%もの冗長なテスト，及び平均43%のテストステートメントを削除することができた

RQ3：Testlerによって再編成されたテストスイートの実行時間はどのくらい短縮しているか？

	Redundancy						Execution Time			Code Coverage				Fault Detection		Analysis Time
	Par. Red. Tests			Red. Test Stmts			Seconds(秒)		%	Stmt		Branch		Mut. Score		CI/MG/TR (Tot)
	Before (#)	Reduced (#)	Reduction (%)	Before (#)	Reduced (#)	Reduction (%)	削除前 Original	削除後 Reorganized	Reduction	Before (%)	Reduced (%)	Before (%)	Reduced (%)	Before (%)	Reduced (%)	Time (seconds)
Collections	110	52	47	210	165	79	17.359	16.440	5	84.7	84.7	72.3	72.3	40.0	40.0	71/70/183 (324)
Math	1,354	946	70	4,052	1,623	40	83.109	72.804	12	92.7	92.7	85.7	85.7	79.0	79.0	101/401/1103 (1,605)
Lang	604	335	55	1,533	563	37	11.053	10.376	6	92.9	92.9	57.5	57.5	66.0	66.0	38/43/372 (453)
Email	21	10	48	38	12	32	7.143	5.933	17	54.6	54.6	50.9	50.9	48.0	48.0	10/12/3 (25)
AssertJ	947	253	27	516	207	40	3.389	2.794	18	95.6	95.6	92.2	92.2	63.0	63.0	244/37/383 (664)
CheckStyle	164	77	47	258	122	47	26.361	21.035	20	95.4	95.4	96.8	96.8	17.0	17.0	172/625/216 (1,013)
PMD	192	122	64	346	203	59	7.550	7.145	5	90.1	90.1	52.8	52.8	25.0	25.0	367/110/32 (508)
LambdaJ	82	42	51	109	55	50	1.468	0.924	37	85.2	85.2	52.2	52.2	85.0	85.0	12/6/13 (31)
Java-library	86	37	43	338	71	21	2.062	1.851	10	96.0	96.0	60.7	60.7	54.0	54.0	58/97/215 (370)
JFreeChart	111	56	50	1,657	374	23	2.752	2.485	10	95.5	95.5	47.6	47.6	24.0	24.0	45/52/205 (302)
Accumulo	289	93	32	683	140	20	65.883	50.794	23	94.7	94.7	71.5	71.5	20.0	20.0	58/333/106 (497)
XmlSecurity	190	156	82	1,102	1,079	98	68.590	57.590	16	78.7	78.7	54.0	54.0	68.0	68.0	42/179/142 (363)
Crunch	78	52	67	125	83	66	161.812	137.657	15	94.5	94.5	74.4	74.4	31.0	31.0	27/123/20 (170)
Tika	75	45	60	141	54	38	78.970	54.350	31	88.4	88.4	57.5	57.5	35.0	35.0	50/673/348 (1,071)
Log4J	336	127	38	711	290	41	261.987	198.343	24	80.5	80.5	38.7	38.7	13.0	13.0	48/490/100 (638)
Total/Average	4,639	2,403	52	11,819	5,041	43	799.488	640.521	20	88.0	88.0	64.3	64.3	44.5	44.5	8,034/536

再編成されたことで，平均20%実行時間を削減することができた

RQ4：Testlerはテストスイートのコードカバレッジと障害検出を保持することができるのか？

	Redundancy						Execution Time			Code Coverage				Fault Detection		Analysis Time
	Par. Red. Tests			Red. Test Stmts			Seconds		%	Stmt		Branch		Mut. Score		CI/MG/TR (Tot)
	Before (#)	Reduced (#)	Reduction (%)	Before (#)	Reduced (#)	Reduction (%)	Original	Reorganized	Reduction	Before (%)	Reduced (%)	Before (%)	Reduced (%)	Before (%)	Reduced (%)	Time (seconds)
Collections	110	52	47	210	165	79	17.359	16.440	5	84.7	84.7	72.3	72.3	40.0	40.0	71/70/183 (324)
Math	1,354	946	70	4,052	1,623	40	83.109	72.804	12	92.7	92.7	85.7	85.7	79.0	79.0	101/401/1103 (1,605)
Lang	604	335	55	1,533	563	37	11.053	10.376	6	92.9	92.9	57.5	57.5	66.0	66.0	38/43/372 (453)
Email	21	10	48	38	12	32	7.143	5.933	17	54.6	54.6	50.9	50.9	48.0	48.0	10/12/3 (25)
AssertJ	947	253	27	516	207	40	3.389	2.794	18	95.6	95.6	92.2	92.2	63.0	63.0	244/37/383 (664)
CheckStyle	164	77	47	258	122	47	26.361	21.035	20	95.4	95.4	96.8	96.8	17.0	17.0	172/625/216 (1,013)
PMD	192	122	64	346	203	59	7.550	7.145	5	90.1	90.1	52.8	52.8	25.0	25.0	367/110/32 (508)
LambdaJ	82	42	51	109	55	50	1.468	0.924	37	85.2	85.2	52.2	52.2	85.0	85.0	12/6/13 (31)
Java-library	86	37	43	338	71	21	2.062	1.851	10	96.0	96.0	60.7	60.7	54.0	54.0	58/97/215 (370)
JFreeChart	111	56	50	1,657	374	23	2.752	2.485	10	95.5	95.5	47.6	47.6	24.0	24.0	45/52/205 (302)
Accumulo	289	93	32	683	140	20	65.883	50.794	23	94.7	94.7	71.5	71.5	20.0	20.0	58/333/106 (497)
XmlSecurity	190	156	82	1,102	1,079	98	68.590	57.590	16	78.7	78.7	54.0	54.0	68.0	68.0	42/179/142 (363)
Crunch	78	52	67	125	83	66	161.812	137.657	15	94.5	94.5	74.4	74.4	31.0	31.0	27/123/20 (170)
Tika	75	45	60	141	54	38	78.970	54.350	31	88.4	88.4	57.5	57.5	35.0	35.0	50/673/348 (1,071)
Log4J	336	127	38	711	290	41	261.987	198.343	24	80.5	80.5	38.7	38.7	13.0	13.0	48/490/100 (638)
Total/Average	4,639	2,403	52	11,819	5,041	43	799.488	640.521	20	88.0	88.0	64.3	64.3	44.5	44.5	8,034/536

（1）テストアサーションはそのまま残る

（2）ステートメントとブランチのカバレッジは保持されている

（3）障害検出率はすべての対象システムについて変わらない

RQ5：Testlerにより各工程にはどの程度時間を要するのか？

	Redundancy						Execution Time			Code Coverage				Fault Detection		Analysis Time
	Par. Red. Tests			Red. Test Stmts			Seconds		%	Stmt		Branch		Mut. Score		CI/MG/TR (Tot)
	Before (#)	Reduced (#)	Reduction (%)	Before (#)	Reduced (#)	Reduction (%)	Original	Reorganized	Reduction	Before (%)	Reduced (%)	Before (%)	Reduced (%)	Before (%)	Reduced (%)	Time (seconds)
Collections	110	52	47	210	165	79	17.359	16.440	5	84.7	84.7	72.3	72.3	40.0	40.0	71/70/183 (324)
Math	1,354	946	70	4,052	1,623	40	83.109	72.804	12	92.7	92.7	85.7	85.7	79.0	79.0	101/401/1103 (1,605)
Lang	604	335	55	1,533	563	37	11.053	10.376	6	92.9	92.9	57.5	57.5	66.0	66.0	38/43/372 (453)
Email	21	10	48	38	12	32	7.143	5.933	17	54.6	54.6	50.9	50.9	48.0	48.0	10/12/3 (25)
AssertJ	947	253	27	516	207	40	3.389	2.794	18	95.6	95.6	92.2	92.2	63.0	63.0	244/37/383 (664)
CheckStyle	164	77	47	258	122	47	26.361	21.035	20	95.4	95.4	96.8	96.8	17.0	17.0	172/625/216 (1,013)
PMD	192	122	64	346	203	59	7.550	7.145	5	90.1	90.1	52.8	52.8	25.0	25.0	367/110/32 (508)
LambdaJ	82	42	51	109	55	50	1.468	0.924	37	85.2	85.2	52.2	52.2	85.0	85.0	12/6/13 (31)
Java-library	86	37	43	338	71	21	2.062	1.851	10	96.0	96.0	60.7	60.7	54.0	54.0	58/97/215 (370)
JFreeChart	111	56	50	1,657	374	23	2.752	2.485	10	95.5	95.5	47.6	47.6	24.0	24.0	45/52/205 (302)
Accumulo	289	93	32	683	140	20	65.883	50.794	23	94.7	94.7	71.5	71.5	20.0	20.0	58/333/106 (497)
XmlSecurity	190	156	82	1,102	1,079	98	68.590	57.590	16	78.7	78.7	54.0	54.0	68.0	68.0	42/179/142 (363)
Crunch	78	52	67	125	83	66	161.812	137.657	15	94.5	94.5	74.4	74.4	31.0	31.0	27/123/20 (170)
Tika	75	45	60	141	54	38	78.970	54.350	31	88.4	88.4	57.5	57.5	35.0	35.0	50/673/348 (1,071)
Log4J	336	127	38	711	290	41	261.987	198.343	24	80.5	80.5	38.7	38.7	13.0	13.0	48/490/100 (638)
Total/Average	4,639	2,403	52	11,819	5,041	43	799.488	640.521	20	88.0	88.0	64.3	64.3	44.5	44.5	8,034/536

コード計測（CI）モデル生成（MG）テストスイート再編成（TR）トータル（Tot）の3つのフェーズに関する実行時間について

Testlerは15件すべてを分析するのに8,034秒（2.2時間）を要した(平均して536秒（8.9分））

まとめ

- 冗長化はテストに対するコストを増やす要因となる
  - 本研究では部分的な冗長性に着目し，軽減する方法を提案した
- Testlerは部分的な冗長性を削除することができた
  - 実行時間についても削減できることがわかった
- Testlerはテストにおいて重要なアサーションを保持することが可能

所感

- テストスメル(test smell)等のエラーとして検出しにくい問題が解決できそう
  - 冗長性の危険性について学ぶことができた
- より依存性が高いテストは冗長性を除去できるのだろうか
  - 依存性にも対応できるとは論文に書いてあるが少し不安要素？