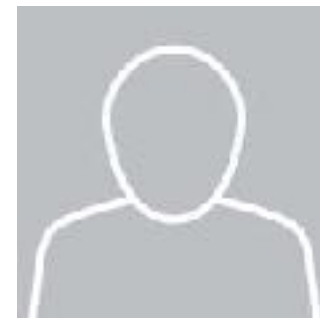


DeFlaker: Automatically Detecting Flaky Tests



Jonathan Bell, Owolabi Legunsen, Michael Hilton, Lamyaa Eloussi,
Tiffany Yung and Darko Marinov

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

George Mason University, University of Illinois at Urbana-Champaign and Carnegie Mellon University

発表者 吉富 楓雅

目次

- 回帰テストとFlaky test
- DeFlaker
- 検証結果
- 結論と所感

要約

Flaky testが存在すると、テストの失敗する原因が回帰によるものなのかが不明となる。また、失敗したテストを再実行するのはコストがかかり、開発サイクルを遅らせる可能性がある。

そこで、再実行せずに自動でFlaky testを検出する技術**DeFlaker**を提案し、様々な再実行アプローチとの比較評価を行う。

回帰テストとFlaky test



回帰テスト

- リポジトリの最新変更が以前の機能を破壊していないかどうかの影響を確認
- 理想的には、右図のように**全て**最新の変更が原因でテストが失敗する
- * 開発者はこれらのエラーに対するデバッグに集中できる

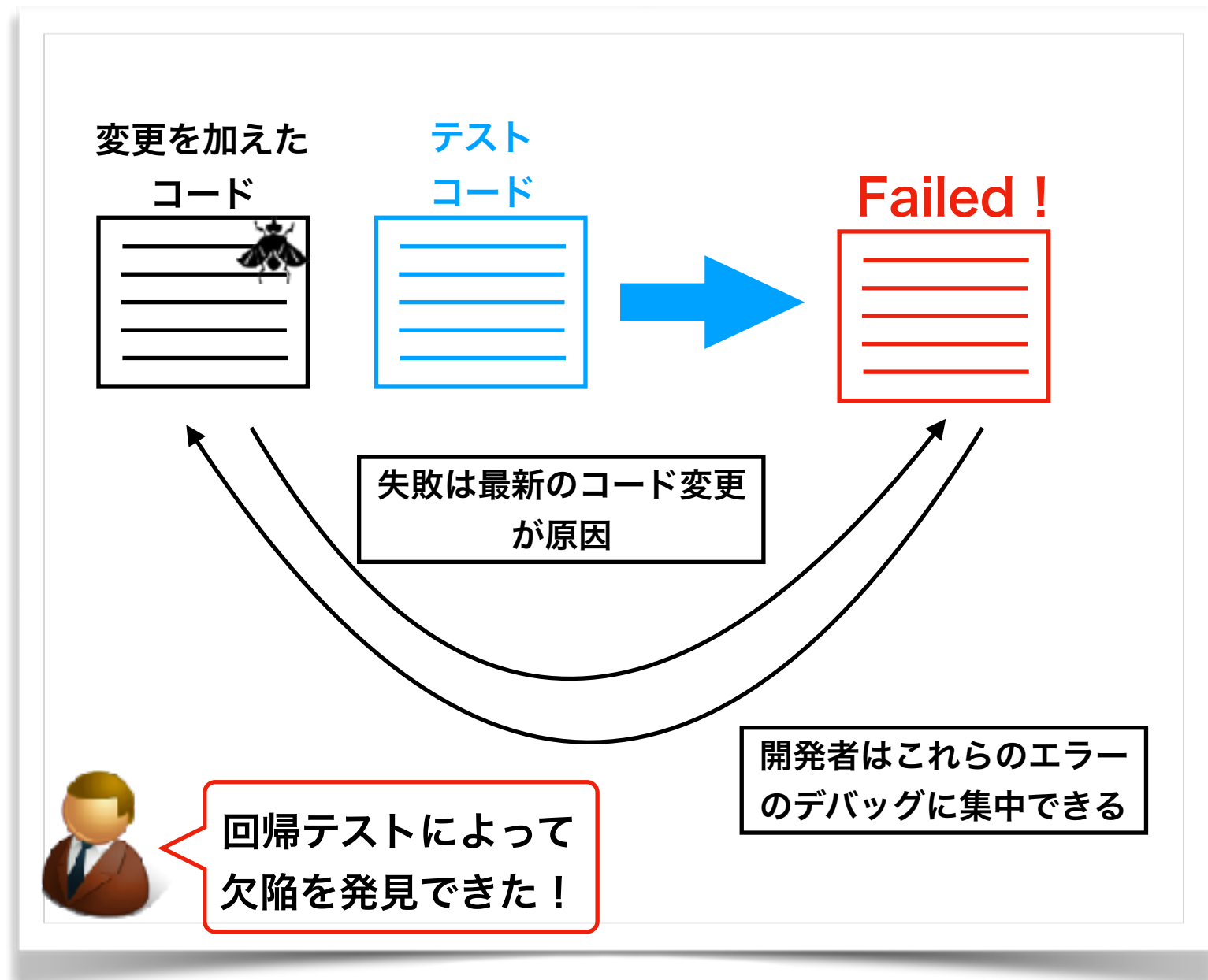


図1：理想的な回帰テスト

Flaky test

- 同じバージョンのコードに実行したテストが成功したり失敗したりする
- テストの不安定性が原因
 - * 開発者はこれらの原因解明に時間とコストをかけるなければならない

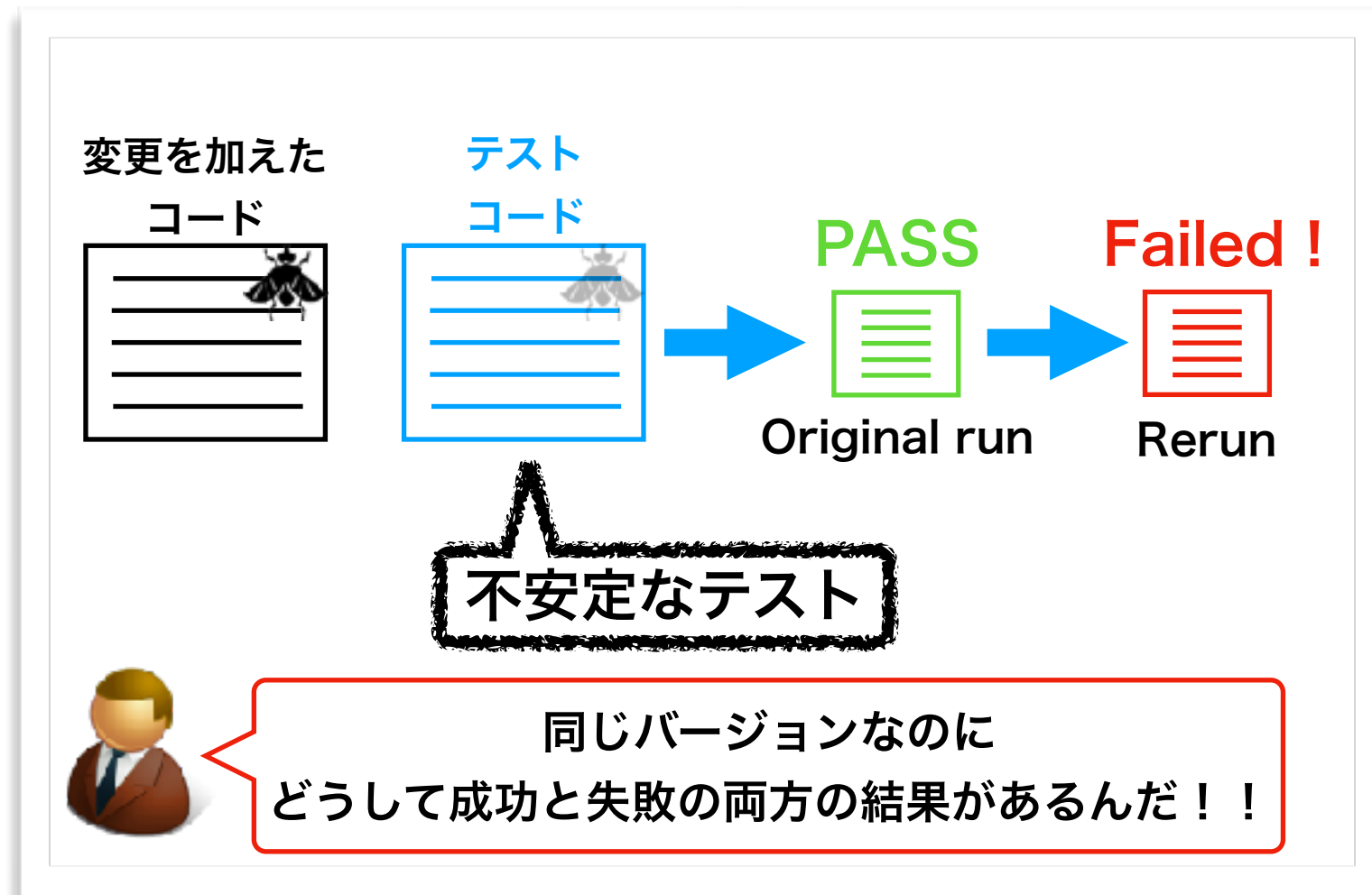


図2：Flaky test

Flaky test

Project
version1



Test



PASS

Project
version1



Test



Failed !

ソースコードに変更なし
definitely flaky!!

Project
version1



Test



PASS

Project
Version2



Test

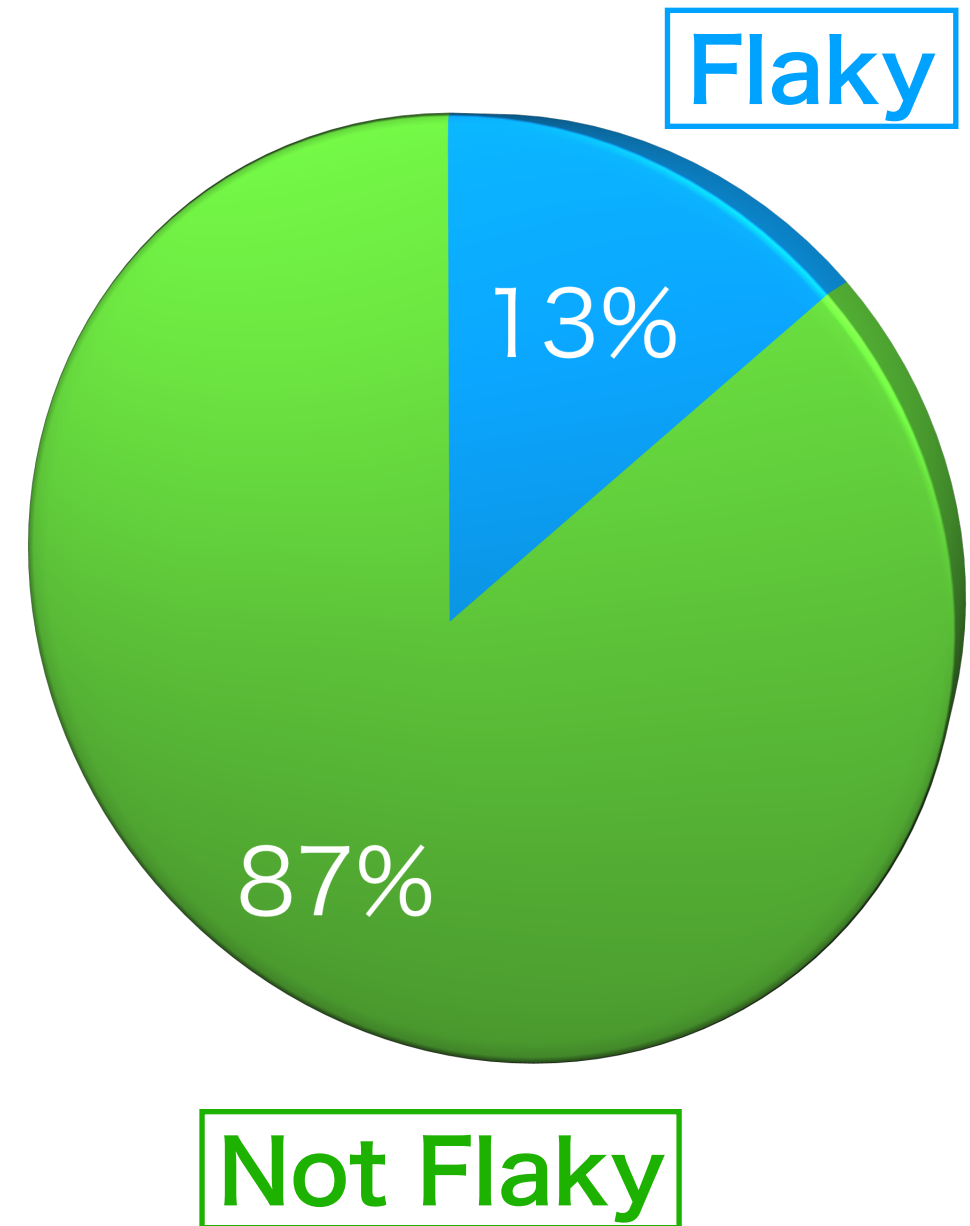


Failed !

ソースコードに変更あり
maybe not...

ビルドツールによるFlaky testの含有率の分析

- Labuschagneらによるビルド失敗におけるFlaky test含有率の分析[1]
- * ビルドツールTravis CI[2]において, 935のビルド失敗のうちFlaky testが原因のものは13%



[1] AdriaanLabuschagne,LauraInozemtseva,andReidHolmes.2017.Measuringthe Cost of Regression Testing in Practice: A Study of Java Projects Using Continuous Integration. In ESEC/FSE.

[2] <https://travis-ci.org>

Flaky testの特定

- Flaky testによるものか, 回帰によるものかを判断するために自動化された手法が必要[3][4]
 - * **Rerun** : フレークテストの失敗を特定するために最も使用されているテクニック
 - ▶ 失敗したことを確認した後に, 失敗した各テストを何度も再実行すること
- Rerunは以下のテストフレームワークでサポートされている
 - * Android, Jenkins, Maven, Spring, Google TAPシステムなど

Rerunの問題点

- 失敗したテストを再実行する方法を説明するものがほとんどない
- Flaky test自体も決定的な定義があるわけではない
 - * Flaky testの再実行が結果を変える保証はない
- 失敗したすべてのテストを再実行する際に、1日あたり何百万から数百万回のテストが失敗するようなことがあれば、非常にコストがかかる
 - * 膨大な計算資源を持つGoogleでさえ、全てのコミットで全てのテストを再実行することはない[5][6]
- 本論文：Rerunを補完する**DeFlaker**の提案

[5]John Micco. 2017. The State of Continuous Integration Testing @Google. (2017). <https://research.google.com/pubs/pub45880.html>.

[6]Celal Ziftci and Jim Reardon. 2017. Who Broke the Build?: Automatically Identifying Changes That Induce Test Failures in Continuous Integration at Google Scale. In ICSE-SEIP.

DeFlaker

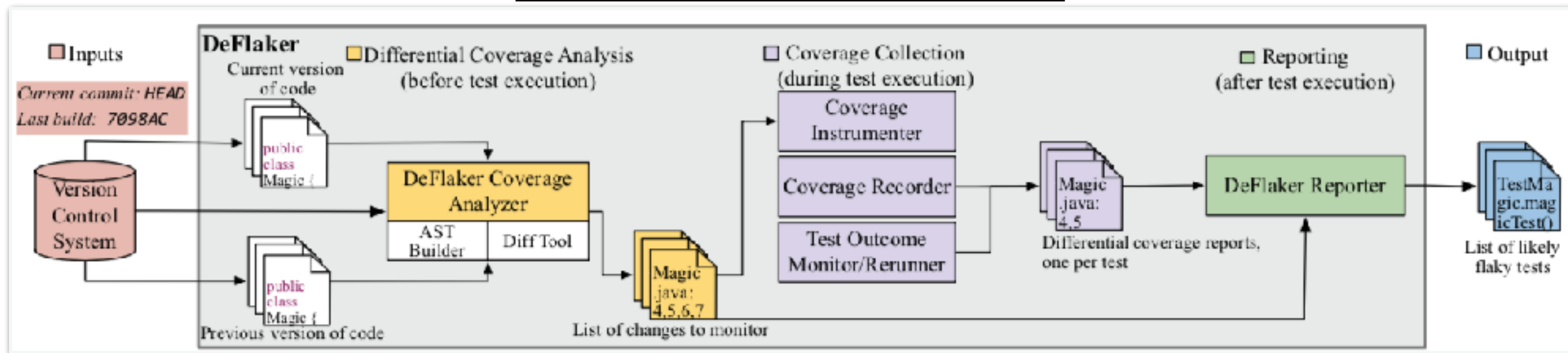


DeFlaker

- ・カバレッジに着目

- ＊ 全てのカバレッジを見るのではなく、変更されたコードのカバレッジのみを収集する(**差分カバレッジ**)

DeFlaker



バージョン管理システム(VCS)から最新バージョンにおけるコード変更を検出

コードを分析し、各変更ファイルのAST(abstract-syntax tree)を構築、
各変更の実行を追跡するために計測をどこに挿入する必要があるかを判断

テストが実行されると変更の実行を監視し、
テストごとに個別の変更カバレッジ報告を生成

リサーチクエスチョン

- DeFlakerがどのような効果をもたらすかについて，以下のリサーチクエスチョンを設けて評価実験する
 - * RQ1：Rerunとは何が異なるのか？
 - * RQ2：DeFlakerは失敗したテストのうち，どのくらいのFlaky testを検出できるのか？
 - * RQ3：DeFlakerはコードの変更をカバーしていないフレークテストのみを検出するが，実際のフレークテストを検出できるのか？
 - * RQ4：DeFlakerをすべてのテストで実行する際，どの程度の負荷がかかるのか？

検証結果



RESULTS

Rerun

- 5,966のビルドのそれぞれを実行し，テストを再実行する
- Flaky testを検出するために以下のようなRerunを用いた方法とDeflakerを比較
 - * **Surefire**：同じ環境のJVM(Java仮想マシン)で最大5回のテストを実行．(Mavenから単体テストを実行するためのプラグイン[7])
 - * **Fork**：Surefireでテストがパスしなかった場合，新しいJVMで最大5回のテスト実行
 - * **Reboot**：テスト間でmvn clean(targetディレクトリの削除)を実行して，最大5回実行の間にマシンを再起動する

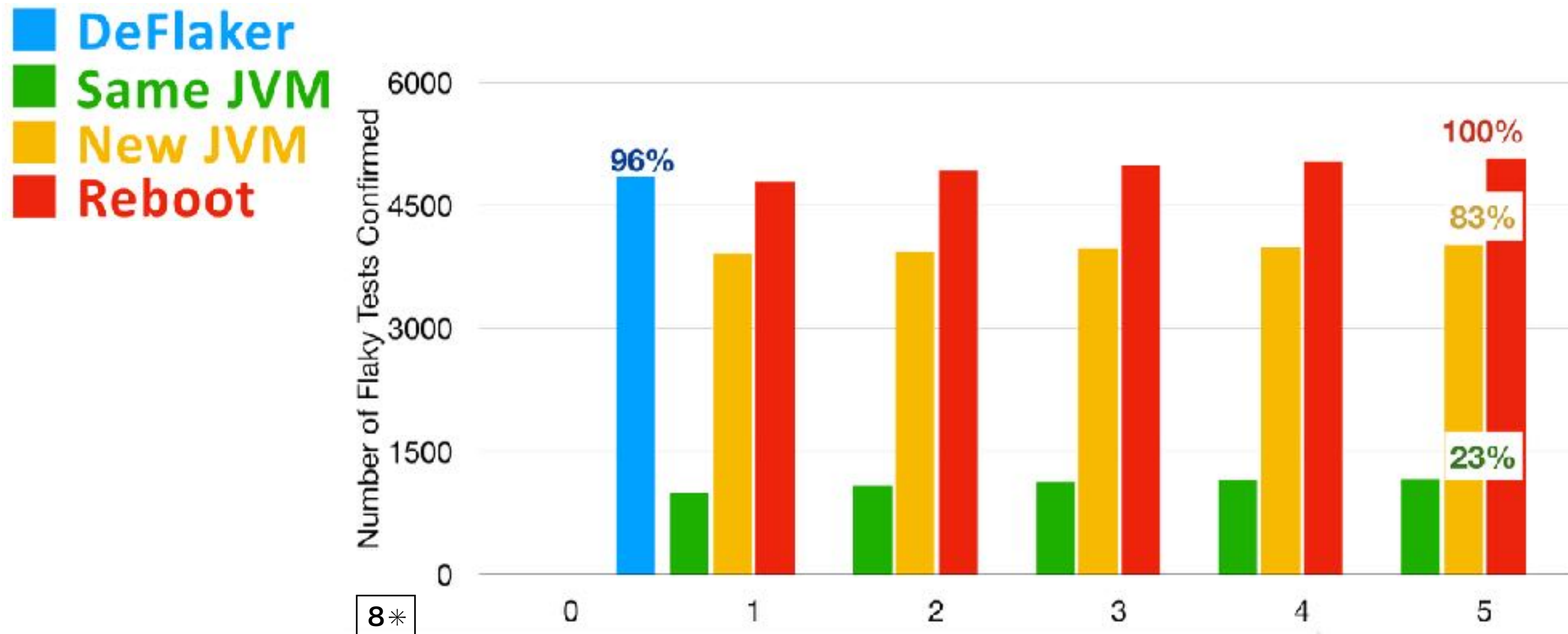
RQ1, RQ2

Project	#SHAs	Test Methods in Project		Total New Failures	Confirmed flaky by RERUN strategy			DeFLAKER labeled as:			
		Total	Failing		Surefire	+Fork	++Reboot	Flaky		Not Flaky	
					Confirmed	Unconf.	Confirmed	Unconf.		Confirmed	Unconf.
achilles	227	337	77	242	13	14	230	225	4	5	8
ambari	500	896	7	75	52	71	74	74	0	0	1
assertj-core	29	6,261	2	3	2	2	2	2	0	0	1
checkstyle	500	1,787	1	1	0	0	0	0	0	0	1
cloudera.oryx	332	275	23	29	5	5	5	5	20	0	4
commons-exec	70	89	2	22	22	22	22	21	0	1	0
dropwizard	298	428	1	60	60	60	60	55	0	5	0
hadoop	298	2,361	365	1,081	284	865	1,054	1,028	25	26	2
handlebars	27	712	7	9	3	7	7	6	2	1	0
hbase	127	431	106	406	62	242	390	383	12	7	4
hector	159	142	12	87	0	74	79	72	4	7	4
httpcore	34	712	2	2	2	2	2	1	0	1	0
jackrabbit-oak	500	4,035	26	34	10	33	34	32	0	2	0
jimfs	164	628	7	21	21	21	21	15	0	6	0
logback	50	964	11	18	18	18	18	18	0	0	0
ninja	317	307	37	122	37	77	110	94	2	16	10
okhttp	500	1,778	129	333	296	305	310	231	0	79	23
oozie	113	1,025	1,065	2,246	42	2,032	2,244	2,234	0	10	2
orbit	227	86	9	86	84	85	85	73	0	12	1
oryx	212	200	38	46	14	14	46	14	0	32	0
spring-boot	111	2,002	67	140	73	107	135	135	3	0	2
tachyon	500	470	4	5	3	5	5	5	0	0	0
togglz	140	227	21	28	5	14	28	28	0	0	0
undertow	7	340	0	0	0	0	0	0	0	0	0
wro4j	306	1,160	114	217	39	96	99	80	8	19	110
zxing	218	415	2	15	15	15	15	15	0	0	0
26 Total	5,966	28,068	2,135	5,328	1,162	4,186	5,075	4,846	80	229	173

Deflakerは
Surefireや
Forkよりも
Flaky testを
検出できている

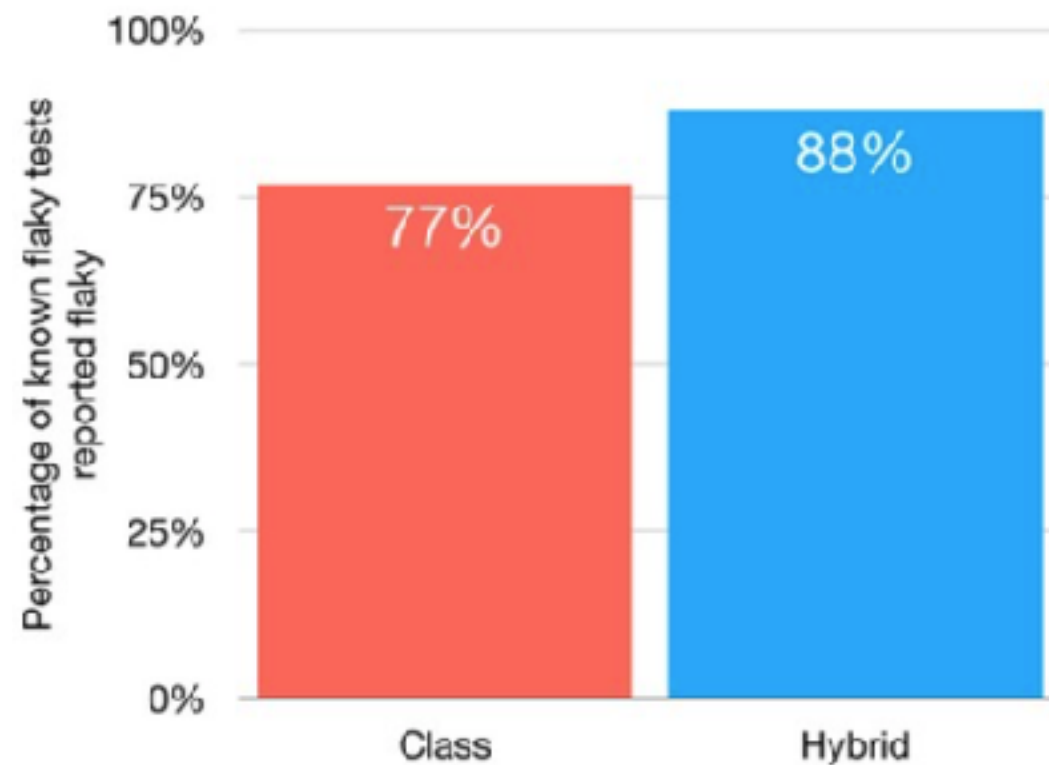
RQ1

Deflakerは再実行することなくほとんどのFlaky testを検出できた



RQ3

- クラスカバレッジを使用する回帰テストツール[9]と比較して、DeFlakerを用いた分析の方がFlaky test発見の正確性が約11%高い

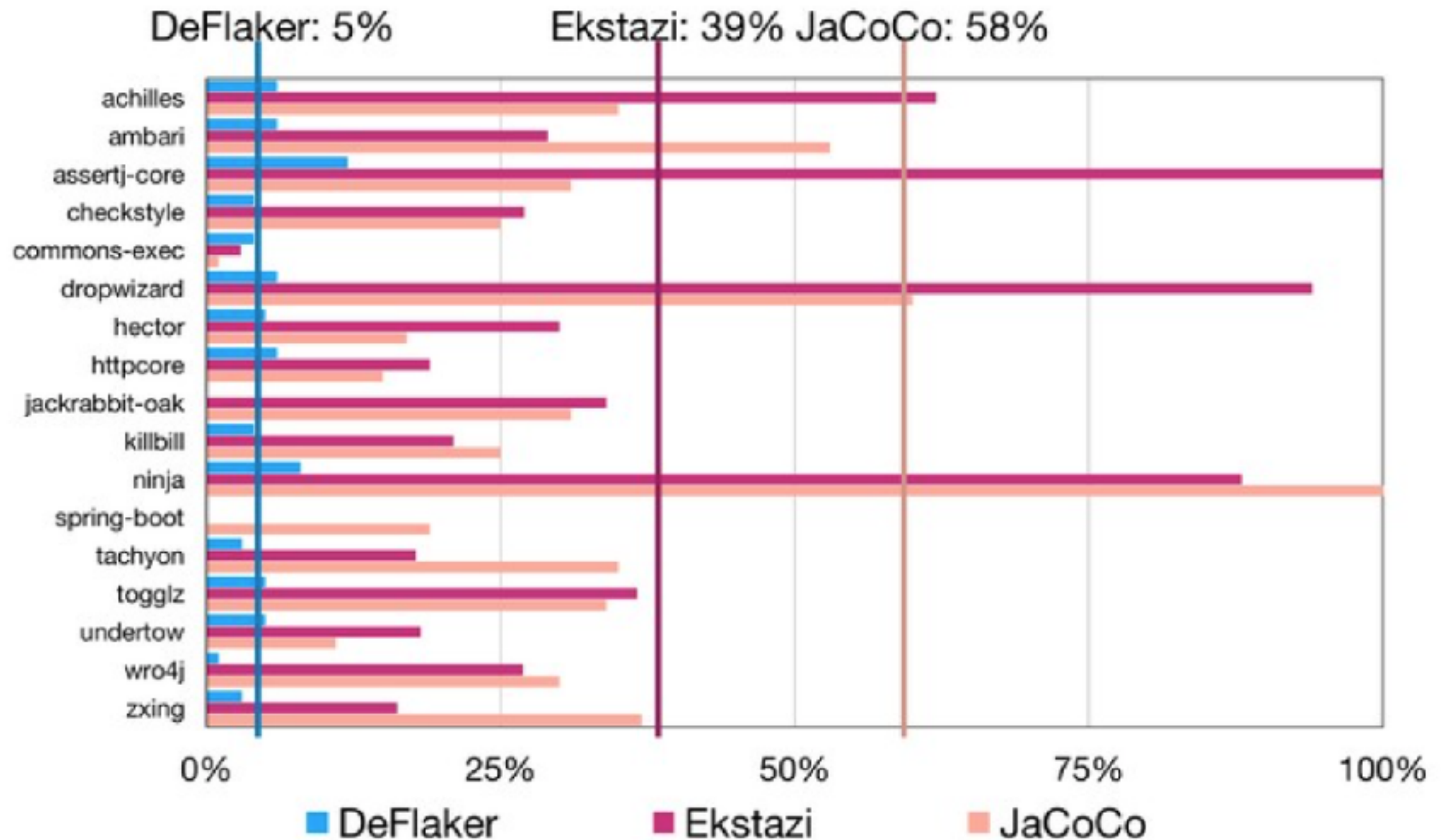


Project	Known Flaky Tests	# SHAs	% of Runs Flaky by:	
			Hybrid Cov	Class Cov
achilles	1	227	77%	71%
ambari	1	500	100%	100%
assertj-core	1	29	97%	83%
checkstyle	1	500	100%	98%
cloudera.oryx	1	332	95%	94%
commons-exec	1	70	94%	69%
dropwizard	1	298	83%	79%
hadoop	3	298	92%	85%
handlebars	7	27	89%	78%
hbase	3	127	86%	72%
hector	1	159	100%	96%
httpcore	1	34	82%	82%
jackrabbit-oak	1	500	75%	66%
jimfs	1	164	54%	24%
logback	11	50	91%	83%
ninja	2	317	90%	80%
okhttp	33	500	91%	81%
oozie	1	113	91%	75%
orbit	1	227	81%	74%
oryx	3	212	100%	100%
spring-boot	5	111	98%	98%
tachyon	7	500	77%	58%
togglz	3	140	97%	94%
undertow	3	7	75%	17%
wro4j	1	306	68%	56%
zxing	2	218	98%	98%
26 Total	96	5,966	88%	77%

Hybrid: 変更されたコードのカバレッジ

RQ4

DeflakerではEkstaziやJacocoに比べて圧倒的に負荷が少ない



結論と所感



結論

- 簡潔に言うと”Deflakerってすごいだる”といった内容の論文
 - * 既存のRerunの手法よりもより実践的
 - ▶ 負荷も少なく Flaky testを検出しやすい

所感

- ☀ 従来の手法と違った視点からツールを作成していてすごい
- ☀ 手軽で負荷も少なく検出できており，実践的である
 - * Rebootは確かに検出は正確であるがコストがかかりすぎる