2020年度 情報知能工学演習第一

Self-Admitted Technical Debt の削除
コンテナ仮想化技術の課題に関する調査

新堂 風
情報知能工学専攻　亀井研究室
2020/12/14 16:40-17:25

アジェンダ

1. 研究背景
2. 文献紹介１
3. 文献紹介２
4. 文献紹介３
5. 今後の展望

研究背景 技術的負債

コード中に存在するバグや解消すべき課題のこと

→ 開発者が理解した上でコードに埋め込まれたものを
Self-Admitted Technical Debt (SATD) と呼ぶ.

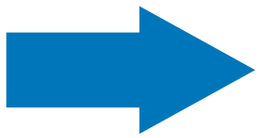
技術的負債の例

```
//TODO is this needed
//Not exact but we cannot do any better
//TODO fix this
//This feels like a hack and it does not work
//FIXME Add flags if possible
```

研究背景 技術的負債

技術的負債の問題

- 解決方法がわからずコードに残ったままにしてしまうものがある
- 将来的にメンテナンスのコストが増える
- ソフトウェアの品質が悪化してしまう



リファクタリングによって解決されるべき

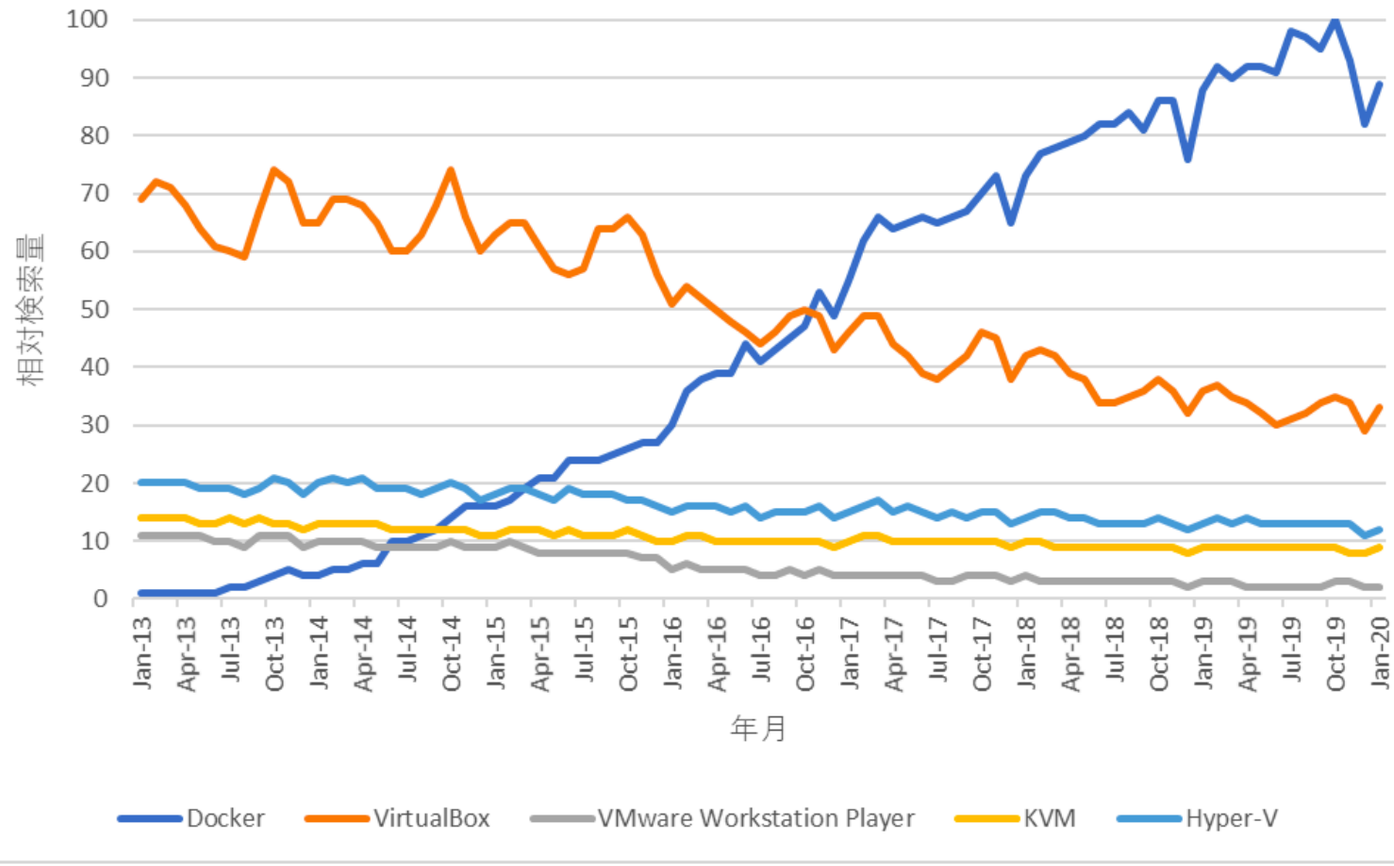
研究背景 コンテナ仮想化技術

アプリ実行環境を構築する一種の仮想化技術

・クラウドやサーバインフラなどで多用されている

・Dockerがデファクトスタンダード

図. Googleにおける仮想化技術の検索量の推移



研究背景 コンテナ仮想化技術

Dockerの仕組み

Dockerfileといわれるファイルに

テキスト形式で実行環境を定義する

メリット

①開発環境を簡単に構築・共有できる

②開発環境や本番環境で同じコンテナを使える

③コンテナを高速で起動できる　etc

Dockerfile

```
# ベースとなるイメージを指定する
FROM ruby:2.5

# コンテナ上のワーキングディレクトリを指定する
WORKDIR /usr/src/

# ディレクトリやファイルをコピーする
# 左側がホストのディレクトリ、右側がコンテナ上のディレクトリ
COPY ./sample.rb /usr/src/sample.rb

# "docker build"時に実行される処理
RUN echo "building..."

# "docker run"実行時に実行される処理
CMD ruby sample.rb
```

図. Dockerfileの例

研究背景 コンテナ仮想化技術

DockerにおけるSATDに着目する理由

- 利便性の高さから多くの開発者に利用されている
- 技術的負債があることのまずさ

  Dockerはコンテナの再利用が基本となっているため、
「あるdockerの負債は，それを再利用する別dockerの問題にも繋がる」

- 比較的新しい技術であるため知見が少ない

研究背景 先行研究

コンテナ仮想化技術における Self-Admitted Technical Debt の調査 [1]

Dockerに含まれるSATDについて

＊ SATDの数や割合

＊ SATDの種類の分布

表. SATDの分類カテゴリと定義

分類名	定義
Code/Workaround	次善策での実装に関する負債
Code/Missing functionality	内部処理の欠如に関する負債
Code/Base image	利用ベースイメージに関する負債
Code/Version	特定バージョンへの固定に関する負債
Test/Integrity	利用バイナリの真正確認に関する負債
Test/Improvement for test	テストの改善に関する負債
Defect/hack	外部システムのバグに関する負債
Defect/latent	潜在バグに関するの負債
Design/Size reduction	イメージのサイズ削減に関する負債
Process/Deployment	デプロイに関する負債
Process/Review	Dockerfile のレビューに関する負債
Unclassifiable	負債ではあるが分類不可な負債

→ 削除についての研究は行われていない

[1] 東英明, 本真佑, 亀井靖高and 楠本真二, ”コンテナ仮想化技術におけるSelf-Admitted Technical Debtの調査” 電子情報通信学会技術研究報告, 202o, pp.25-30.

文献紹介

文献紹介

1. SATD 削除期間や割合についての調査
2. SATD 削除時のソースコードへの影響についての調査
3. Docker におけるビルドの失敗に関する調査

”An Empirical Study on the Removal of Self-Admitted Technical Debt” [2]

５つの大規模なオープンソースプロジェクトのJavaファイルにおける
SATDの削除についての研究

→ 先行研究でSATDの削除がどのように調査され、どのような結果が示されているか

[2] E.D.S.Maldonado, R.Abdalkareem, E.Shihab and A.Serebrenik, ”An Empirical Study on the Removal of Self-Admitted Technical Debt,” 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, 2017, pp. 238-248, doi: 10.1109/ICSME.2017.8.

文献紹介

1. SATD 削除期間や割合についての調査
2. SATD 削除時のソースコードへの影響についての調査
3. Docker におけるビルドの失敗に関する調査

”Was Self-Admitted Technical Debt Removal a Real Removal? An In-Depth Perspective” [3]

文献 [2] で対象にしたデータをもとに

SATD 削除時のコードの変化や削除手法についての調査

→ 先行研究でSATDの削除手法がどのように調査され、どのような結果が示されているか

[3] F. Zampetti, A. Serebrenik and M. Di Penta, ”Was Self-Admitted Technical Debt Removal a Real Removal? An In-Depth Perspective,” 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), Gothenburg, 2018, pp. 526-536.

文献紹介

1. SATD 削除期間や割合についての調査
2. SATD 削除時のソースコードへの影響についての調査
3. Docker におけるビルドの失敗に関する調査

” An Empirical Study of Build Failures in the Docker Context ” [4]

3,828件のGithubプロジェクトをもとに

Dockerのビルドの失敗の頻度やその修正に関する調査

→ Dockerにおいてのバグの数やその修正にどれくらいの労力がかかるのかについて

[4] Yiwen Wu, Yang Zhang, Tao Wang, and Huaimin Wang. 2020. An Empirical Study of Build Failures in the Docker Context. In Proceedings of the 17th International Conference on Mining Software Repositories (MSR ’20). Association for Computing Machinery, New York, NY, USA, 76–80. DOI : https://doi.org/10.1145/3379597.3387483

文献1

文献1 SATD 削除期間や割合についての調査

従来の研究では
ソフトウェアに悪影響を及ぼすことが主張されてきたが、

SATDはプロジェクトに長期間（最大10年）存在することもある

→ 削除についての調査をすることで
返済の必要がない安全なSATDのパターンを明らかにできる

文献1　SATD 削除期間や割合についての調査

SATD についての Research Question

RQ1:　どの程度削除されるのか?

RQ2:　削除するのは誰か？

RQ3:　どのくらいの期間プロジェクト内で存続するか？

RQ4:　削除はどのような活動によって行われるのか？

文献1 SATD 削除期間や割合についての調査

データ収集・前処理

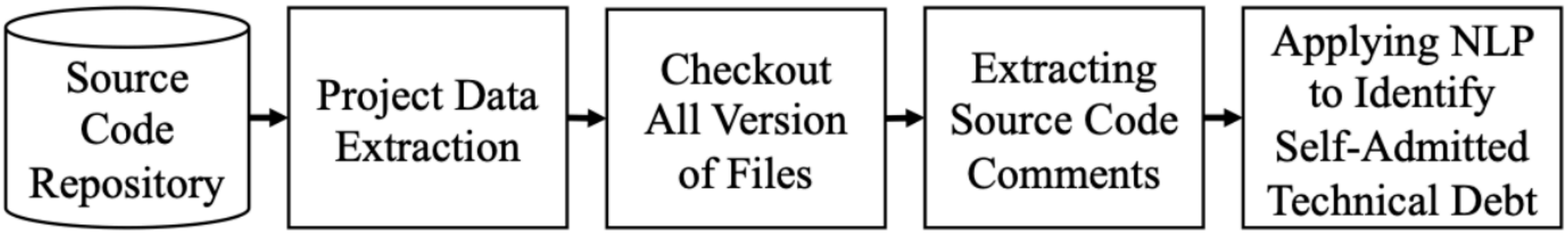


図. [文献1] データ処理の手順

文献1 SATD 削除期間や割合についての調査

Source Code Repository

5つのオープンソースプロジェクト

Camel / Gerrit / Hadoop / Log4j / Tomcat

プロジェクトの選定

- 異なる応用領域・規模
- コメントが多い
- 活動性が高い

文献1 SATD 削除期間や割合についての調査

Project Data Extraction

・30,915件のファイル

・7,749,969件のコメント

表. 対象プロジェクトの詳細

Project	Project details				Comments details			
	# Java files	SLOC	# file versions	# contributors	# comments	# comments after filtering	# TD comments	# unique TD comments
Camel	15,091	800,488	254,920	289	1,634,361	700,412	20,141	4,331
Gerrit	3,059	222,476	53,298	270	1,018,006	129,023	4,810	271
Hadoop	8,466	996,877	79,232	160	2,512,673	1,172,051	18,927	1,164
Log4j	1,112	30,287	12,609	35	248,276	61,690	1,893	135
Tomcat	3,187	297,828	46,716	32	2,336,653	1,081,492	26,725	1,317

文献1 SATD 削除期間や割合についての調査

Checkout All Version of Files

・SATDを含む最初に利用可能なバージョン：　**導入日**

・SATDコメントが削除、ファイルが削除された日：　**削除日**

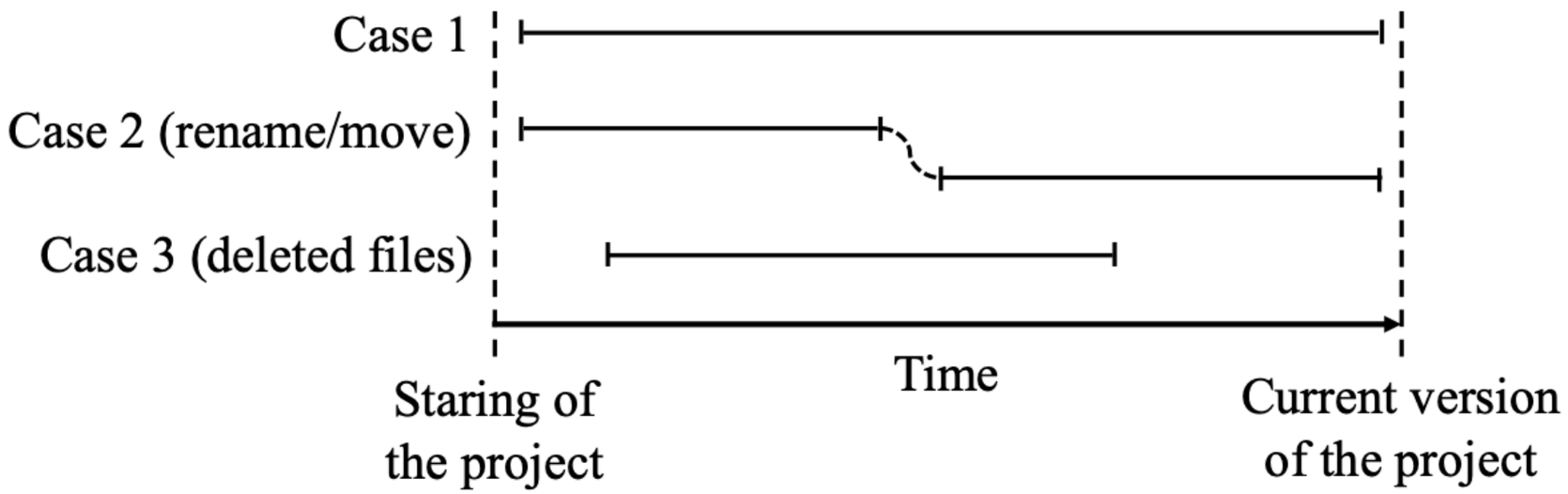


図. ファイルのバージョンについての異なるケース

文献1 SATD 削除期間や割合についての調査

Extracting Source Code Comments

・オープンソースライブラリ SrcML [5]を用いて コメント・関連情報を抽出

・明らかにSATDではないコメントを除外

　・自明の負債を含まないライセンスコメント

　・コメントアウトされたソースコード

　・IDEによって自動的に生成されるコメント

　・Javadoc のコメント

コメントの

53.3%〜87.3%を削減

表. 対象プロジェクトの詳細

Project	Project details				Comments details			
	# Java files	SLOC	# file versions	# contributors	# comments	# comments after filtering	# TD comments	# unique TD comments
Camel	15,091	800,488	254,920	289	1,634,361	700,412	20,141	4,331
Gerrit	3,059	222,476	53,298	270	1,018,006	129,023	4,810	271
Hadoop	8,466	996,877	79,232	160	2,512,673	1,172,051	18,927	1,164
Log4j	1,112	30,287	12,609	35	248,276	61,690	1,893	135
Tomcat	3,187	297,828	46,716	32	2,336,653	1,081,492	26,725	1,317

[5] M. L. Collard, M. J. Decker, and J. I. Maletic. srcML: An infrastructure for the exploration, analysis, and manipulation of source code: A tool demonstration. In 29th International Conference on Software Maintenance, pages 516–519. IEEE Computer Society, 2013.

文献1 SATD 削除期間や割合についての調査

Applying NLP to Identify SATD

Maldonadoら

“Using natural language processing to automatically detect self-admitted technical debt” [6] によって提供されたデータを用いて訓練した

NLP分類器を用いてSATDコメントを抽出

[6] E. Maldonado, E. Shihab, and N. Tsantalis. Using natural language processing to automatically detect self-admitted technical debt. *IEEE Transactions on Software Engineering*, page to appear, 2017

文献1 SATD 削除期間や割合についての調査

RQ1: SATDはどの程度削除されるのか?

SATDの大部分（平均74.4%）が削除されている。

→ 開発者は「SATDを認識し、気にかけている傾向」がある

表. プロジェクトごとのSATDの詳細

Project	# Identified	# Removed	% Removed	% Remaining
Camel	4,331	3,926	90.6	9.4
Gerrit	271	208	76.7	23.3
Hadoop	1,164	472	40.5	59.5
Log4j	135	118	87.4	12.6
Tomcat	1,317	1,009	76.6	23.4
Average	-	-	74.4	25.6
Median	-	-	76.7	23.3

文献1 SATD 削除期間や割合についての調査

RQ2: SATDを削除するのは誰か？

SATDの**大部分（平均54.4%）**が SATDを導入した本人により削除されている。

Hadoop は 24.6% と低く，外れ値になる傾向がある

開発者の離職率が高かったり、技術的な負債に対処するためのプロセスが不足していたりするなど、多くの理由が考えられる

表. プロジェクトごとのSATDの自己削除

Project	# Removed	# Self-removed	% Self-removed
Camel	3,926	2,652	67.5
Gerrit	208	149	71.6
Hadoop	472	116	24.6
Log4j	118	72	61.0
Tomcat	1,009	578	57.3
Average	-	-	54.4
Median	-	-	61.0

文献1 SATD 削除期間や割合についての調査

RQ3: SATDはどのくらいの期間プロジェクト内で存続するか？

SATDがプロジェクト内に留まる時間

＊平均値

82日 ～ 613.2日

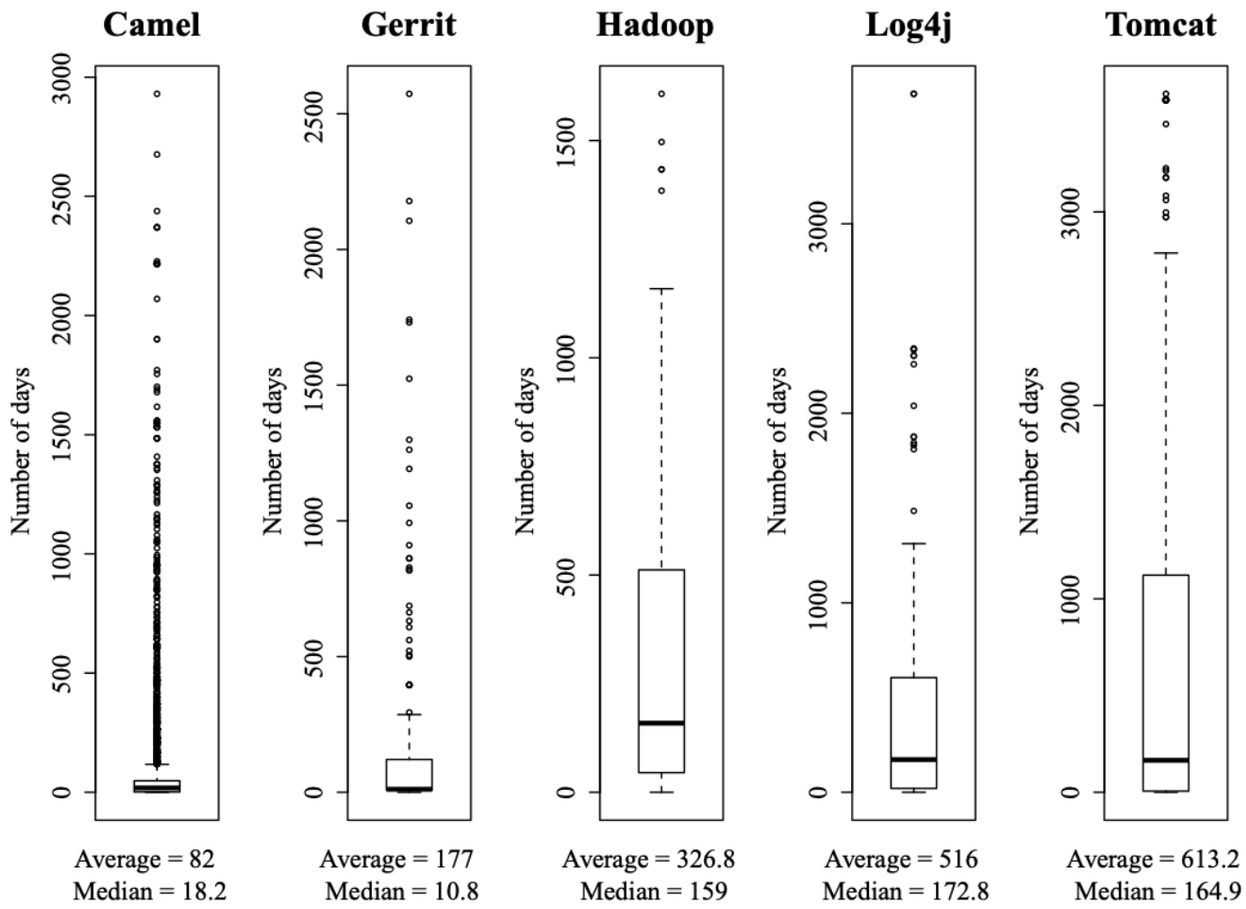


図. SATDがプロジェクト内に留まる時間の分布

文献1 SATD 削除期間や割合についての調査

RQ3: SATDはどのくらいの期間プロジェクト内で存続するか？

最初の数百日で急落している

→ 重要なSATDが急速に解消されている

プロジェクトによって

急落の度合い・フラットになる箇所が異なる

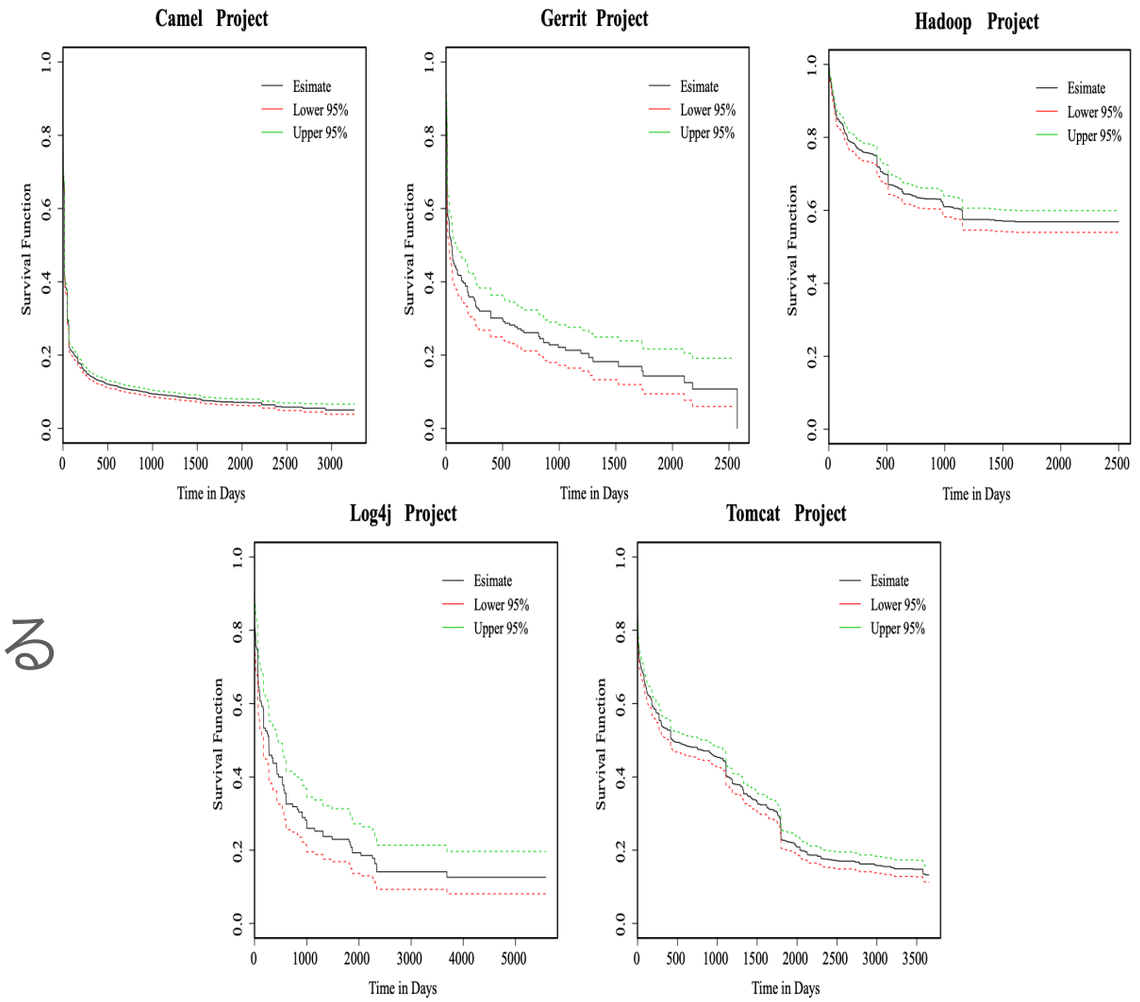


図. 各プロジェクトにおけるSATDの生存プロット

文献1　SATD 削除期間や割合についての調査

RQ3: SATDはどのくらいの期間プロジェクト内で存続するか？

自己削除されたSATDは、自己削除されていないSATDよりも早く削除される

→ 統計的に、２種類の削除に差があることが示されている

図. SATDの削除期間の分布（自己削除 vs 非自己削除）

文献1 SATD 削除期間や割合についての調査

RQ4: SATDの削除はどのような活動によって行われるのか？

５つのプロジェクトと ApacheAnt , Jmeter の２つのプロジェクト

→ 負債を追加・削除した 250人のうち188人にアンケート送信し、14人が回答

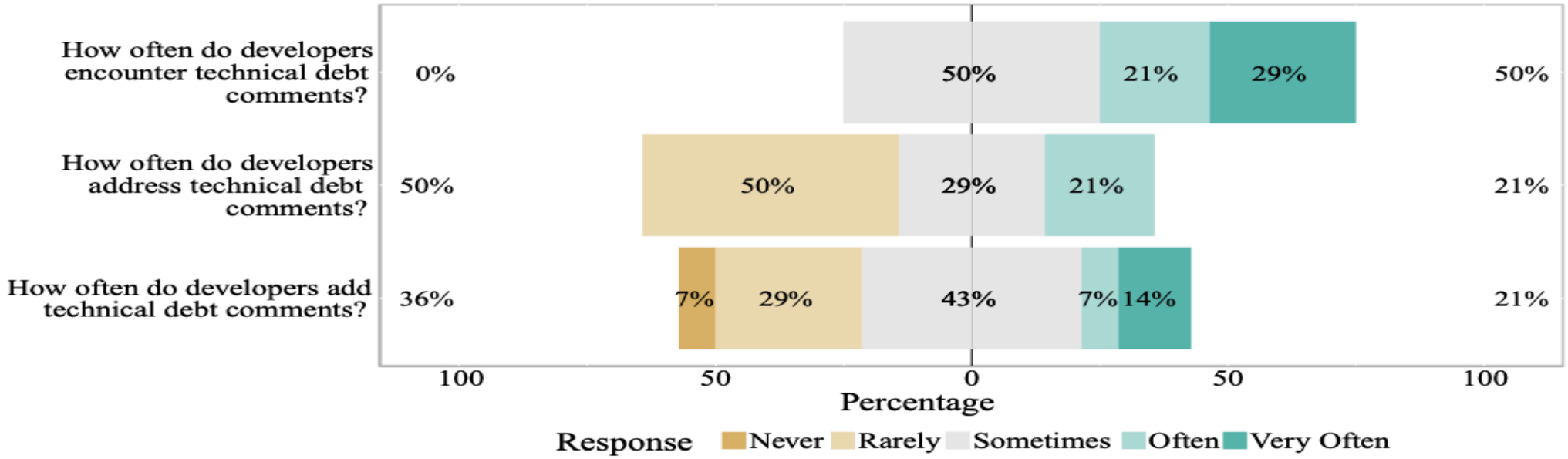


図. 開発者のSATDに関するアンケート調査結果

文献1 SATD 削除期間や割合についての調査

RQ4: SATDの削除はどのような活動によって行われるのか？

・潜在的なバグや改善のための文書化、ソース内の目印としてSATDを追加する
→ 削除の必要性について認識している

・開発者は、タスクやプロジェクトに応じた時間的なプレッシャーにより、SATDを追加することが多い。

・ バグを修正時・新機能追加時に、SATDを削除する
→ リファクタリングやコード改善の一環としてSATDを削除することはほとんどない

文献1 SATD 削除期間や割合についての調査

まとめ

- 大部分は（平均74.4 %）削除されている
- 平均54.4 %でSATDを導入した本人により削除されている
- 削除されるまでの期間は平均82〜613.2日である
- 開発者はSATDを削除する必要性を認識しているが

  そのための正式なプロセスはなく、ほとんどはバグ修正の一環として削除されている

→ **効果的・体系的にSATDに対処できるような技術が必要である**

文献2

文献2 SATD 削除時のソースコードへの影響についての調査

文献1

→ SATDの削除について主に定量的な観点をもとに調査

SATDの削除手法の詳細な調査を行うことで

→ 特定の種類のSATDに対応するためのパターン学習や

開発者へ開発手法の知見の提供を行うことが可能になる

文献2 SATD 削除時のソースコードへの影響についての調査

SATD の削除手法 についての Research Question

RQ1:　削除された際の手法はどのように分類されるか？

RQ2:　削除はコミットメッセージに反映されているか？

RQ3:　削除時にコードにどのような変更が行われているか？

文献2 SATD 削除時のソースコードへの影響についての調査

データ収集・前処理（文献1のデータを用いる）

・重複や不整合データの除外

・メソッドレベルのSATDに着目

　・ファイルやクラス全体のSATDを無視

表. 対象プロジェクトの統計値

Project	# SATD	# SATD Removals	# Attached to method	% Removals
Camel	1,282	877	748	58.35
Gerrit	150	88	88	58.67
Hadoop	998	306	277	27.76
Log4j	113	96	93	82.30
Tomcat	1,184	876	777	65.63

文献2 SATD 削除時のソースコードへの影響についての調査

データ収集・前処理（文献1のデータを用いる）

・SATDの削除に着目するため、以下のタイプのSATDを除外

・クラス名の変更がSATDの削除と判定されている場合

・SATDがコード内の別の場所に移動されている場合

・構文的に削除された後もSATDを表している場合

表. SATD削除のフィルタリング結果

Project	Class Renamed	Comment Moved	Comment Changed still SATD	SATD comments
CAMEL	0	12	98	638
GERRIT	0	12	5	71
HADOOP	4	10	26	237
LOG4J	19	4	7	63
TOMCAT	0	7	88	682

文献2 SATD 削除時のソースコードへの影響についての調査

データ収集・前処理（文献1のデータを用いる）

・SATDの削除とコミットメッセージの関連調査

ｉ）コサイン類似度を利用し、類似度を調査

ii）類似しているものを、手動で５段階ラベル付け

(1)全く似ていない ～ (5)非常に似ている

反映されている例

CommitMessage：**“Implement a TODO: Log receipt of an unexpected ACK”**

SATD：　**“// Unexpected ACK. Log it? //TODO”**

文献2 SATD 削除時のソースコードへの影響についての調査

RQ1: 削除された際の手法はどのように分類されるか？

・**２・３列目：** SATD解決のためではなく、「偶然」削除されている
・**４列目：** SATDを受け入れている or システムの進化により問題がなくなっている
・**５列目：** SATDへの対応を目的としている可能性が高いもの

表. SATD削除方法の種類

Project	Class Removal	Method Removal	Method not Changed	Method Changed	Total
Camel	60 (9%)	105 (16%)	109 (17%)	364 (57%)	638
Gerrit	14 (20%)	9 (13%)	3 (4%)	45 (63%)	71
Hadoop	63 (27%)	39 (16%)	13 (5%)	122 (52%)	237
Log4j	29 (46%)	9 (14%)	1 (2%)	24 (38%)	63
Tomcat	334 (49%)	74 (11%)	50 (7%)	224 (33%)	682

文献2 SATD 削除時のソースコードへの影響についての調査

RQ2: 削除はコミットメッセージに反映されているか？

コサイン類似度が 0.3以上の148件について手動で分類

→ 削除全体のうち **8%(131件)** がコミットメッセージに反映

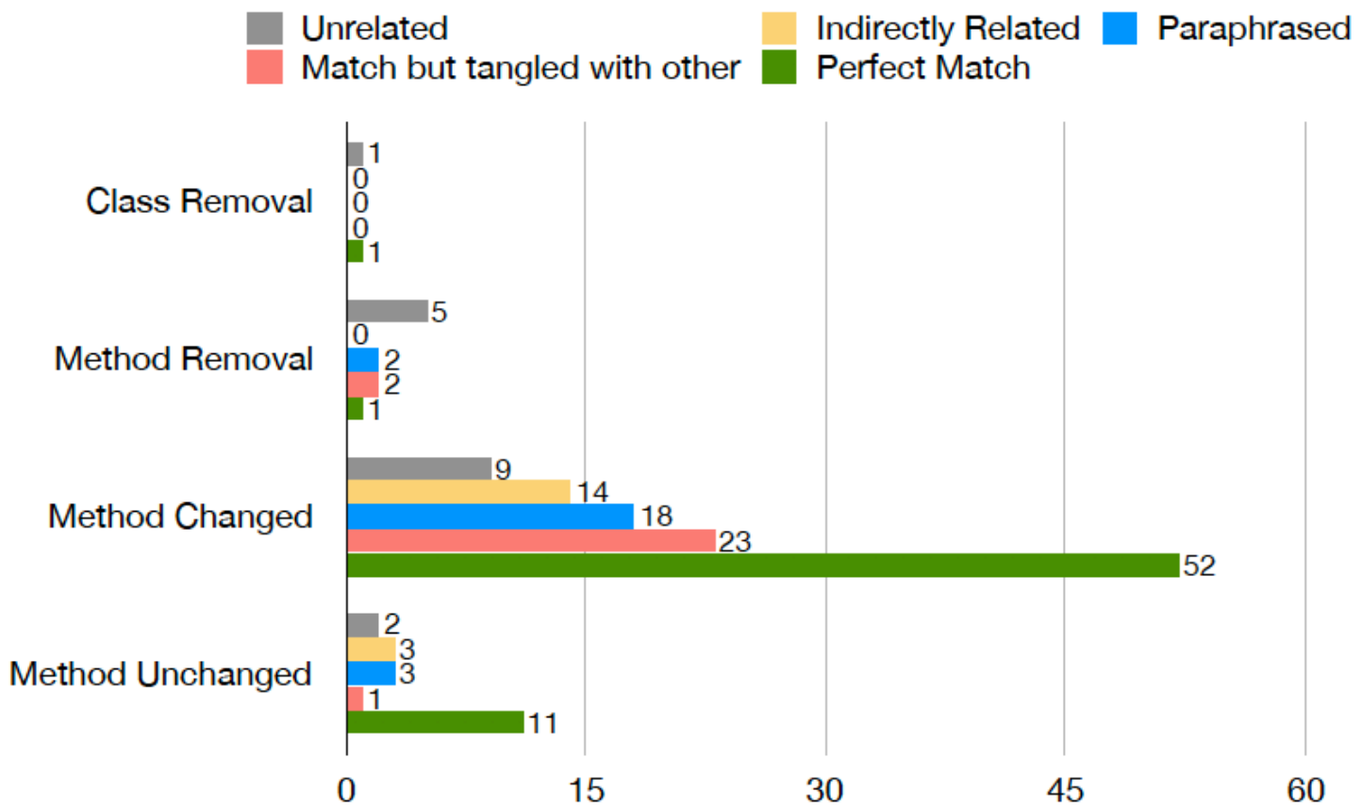


図. 異なる削除タイプにおけるコミットメッセージとの関連度

文献2 SATD 削除時のソースコードへの影響についての調査

RQ3: 削除時にコードにどのような変更が行われているか？

・メソッドが変更されているSATDについて 変更の種類を分類

→ 複数の変更が行われることがあるため合計は 100%ではない

表. SATD削除におけるソースコードの変化

Project	Add/Remove Method Calls	Add/Remove Conditionals	Add/Remove Try-Catch	Modify Method Signature	Modify Return	Other	Total
Camel	165 (45%)	61 (17%)	9 (3%)	36 (10%)	15 (4%)	145 (40%)	364
Gerrit	16 (36%)	8 (18%)	3 (7%)	3 (7%)	3 (7%)	23 (51%)	45
Hadoop	42 (34%)	13 (11%)	2 (2%)	6 (5%)	4 (3%)	67 (55%)	122
Log4j	8 (33%)	7 (29%)	0	0	0	10(42%)	24
Tomcat	59 (26%)	59 (26%)	5 (2%)	17 (8%)	7 (3%)	94 (42%)	224
Total	290	148	19	62	29	339	779

※その他：変更が複雑すぎるもの

文献2 SATD 削除時のソースコードへの影響についての調査

RQ3: 削除時にコードにどのような変更が行われているか？

・条件文について、その種類の分布を調査

→　if文の変更によるSATDの削除が多い

表. SATD削除における　条件文の変化の詳細

Project	If	Loop	Switch	Total
Camel	55	7	2	61
Gerrit	7	2	0	8
Hadoop	11	2	1	13
Log4j	7	0	1	7
Tomcat	56	5	1	59

文献2 SATD 削除時のソースコードへの影響についての調査

まとめ

- SATDの削除は、クラスやメソッド全体が削除された時に

  「偶発的」に発生している

- 削除がコミットメッセージに反映されているケースは わずか 8%

- 開発者は複雑な変更だけでなく、

  メソッドの呼び出しや条件式を変更する傾向がある

→ 知見をもとに特定のSATDに対応するための変更パターンの学習、
パターンに基づいた 開発者への推奨事項を提供していくことができる

文献3

文献3 Docker におけるビルドの失敗に関する調査

Dockerでは、頻繁にビルドが失敗することがあり、
その修正には多くの労力が必要になる

従来の研究
→ 大規模な開発でのビルドの失敗率についての調査

ビルドの失敗とその修正についての調査を行うことで
ビルドの効率を向上させるプロセス改善や開発の指針に繋がる

文献3 Docker におけるビルドの失敗に関する調査

Docker のビルド についての Research Question

RQ1:　ビルドはどのくらいの頻度で失敗するか？

RQ2:　ビルドを修正するのにどれくらい時間がかかるか？

RQ3:　失敗の頻度と修正時間は
　　時間の経過とともにどのように変化するか？

文献3 Docker におけるビルドの失敗に関する調査

データ収集・前処理

Docker/DockerHubAPIを利用している Githubプロジェクト

→ **3,828件**のプロジェクト / **870,580件**のビルドのデータ

※ ビルド数が10以下のものをフィルタリング

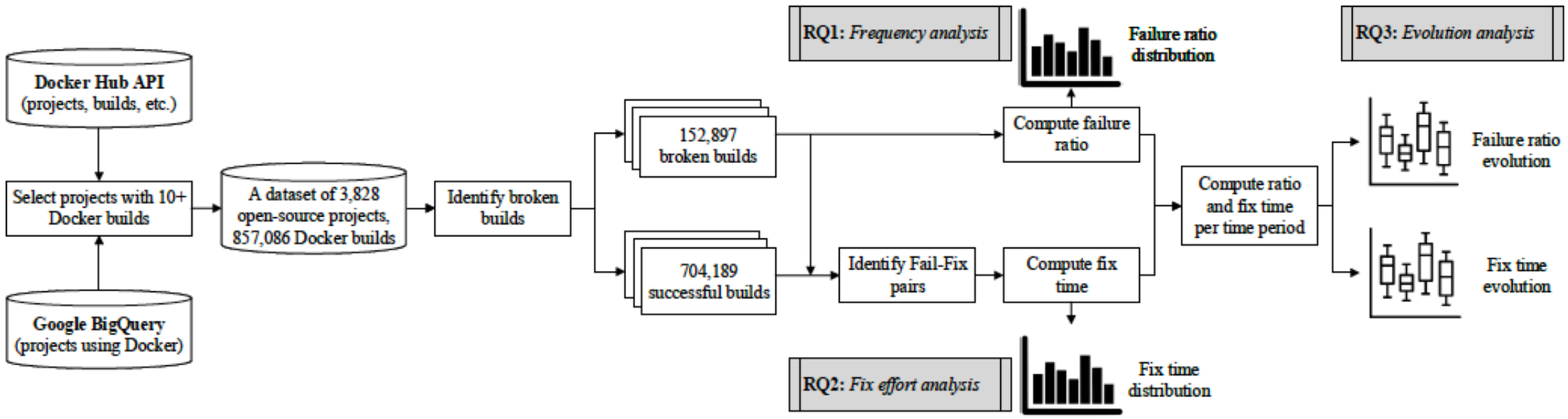


図. [文献3] データ処理の手順

文献3 Docker におけるビルドの失敗に関する調査

データ収集・前処理

データの統計

表. プロジェクトごとのビルド数の基本統計

Statistic	Mean	St.Dev.	Min	Median	Max
#Total builds	224	883.9	10	36	30,615
#Successful builds	184	785.5	0	28	30,270
#Broken builds	40	245.4	0	5	10,500

文献3 Docker におけるビルドの失敗に関する調査

データ収集・前処理

ビルドの失敗とその修正のコミットペアの取得 （有効なペア10,566件）

以下の手順でプロジェクトのビルドシーケンスを反復処理

１）ビルドが成功した直後に続く壊れたビルドを発見(A)

２）成功したビルド(B)に遭遇するまで壊れたビルドをスキップ

※ 開発者のスケジュールと修正時間の複合化を避けるために

　修正時間が12時間を超えるものをフィルタリング

文献3 Docker におけるビルドの失敗に関する調査

RQ1: ビルドはどのくらいの頻度で失敗するか？

従来の研究　(google)

C++ ビルド失敗の中央値 37.4% ・ Java ビルド失敗の中央値 29.7%

文献3

568プロジェクト (14.8%)

→ ビルド失敗率　０％

2,055プロジェクト (53.7%)

→ ビルド失敗率 20%未満

1,205プロジェクト (31.5%)

→ ビルド失敗率 20%以上

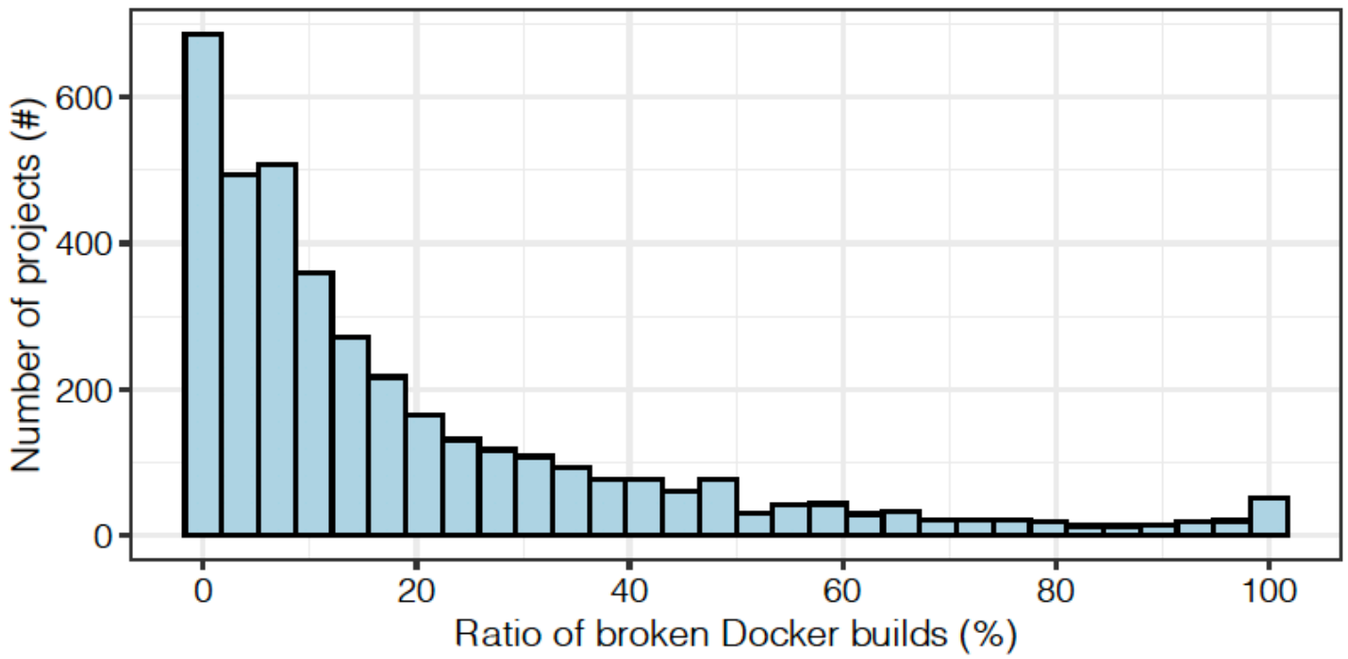


図. Dcokerビルドの失敗率の分布

文献3 Docker におけるビルドの失敗に関する調査

RQ1: ビルドはどのくらいの頻度で失敗するか？

ビルド数とビルド失敗率の相関関係

ビルド頻度の高いプロジェクトは
壊れたビルドの割合が比較的低く、統計的に有意である

表. ビルドの失敗率と件数

ビルド失敗率	ビルド数 中央値
0%	16ビルド
20%未満	58ビルド
20%以上	31ビルド

文献3 Docker におけるビルドの失敗に関する調査

RQ2: ビルドを修正するのにどれくらい時間がかかるか？

- 511件 (4.8%) : 1分未満
- 2,105件 (19.9%) : 1〜10分
- 4,511件 (42.7%) : 10〜100分
- 3,431件 (32.5%) : 100分以上

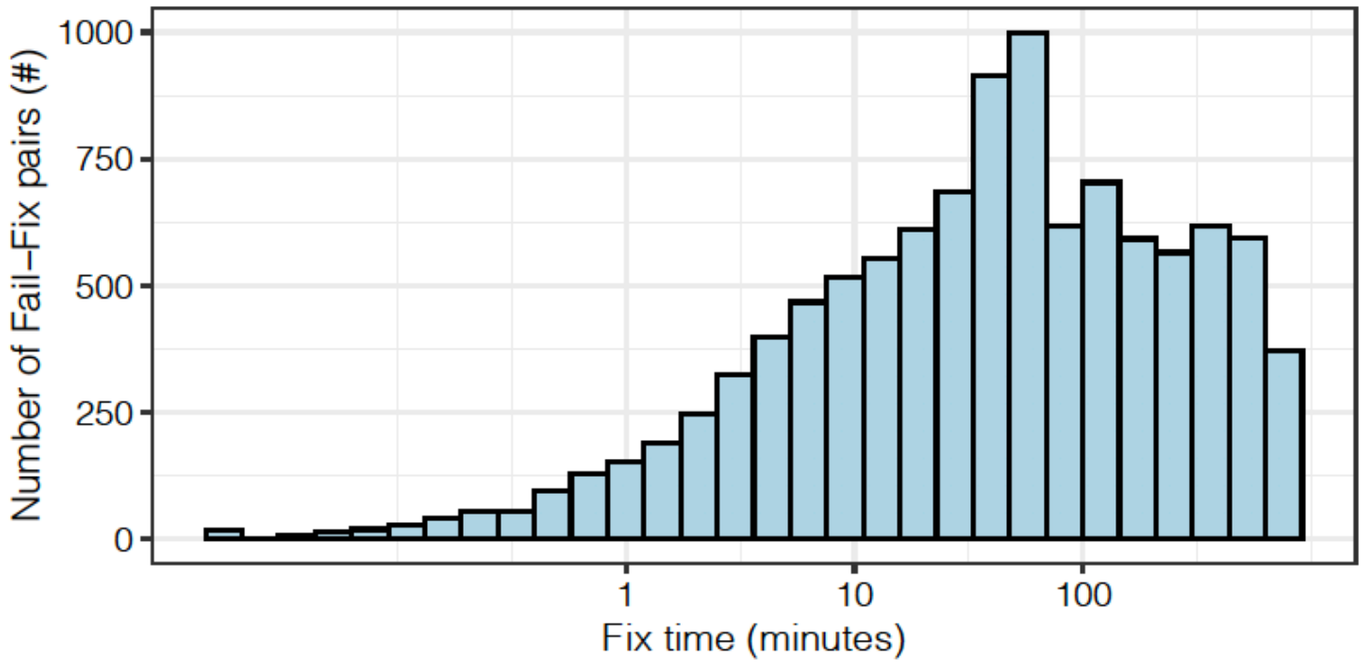


図. Dockerビルドの修正時間の分布

文献3 Docker におけるビルドの失敗に関する調査

RQ2: ビルドを修正するのにどれくらい時間がかかるか？

ビルド修正時間の中央値：44.2分
従来の研究 (Google[7]) ：12分未満

- Google が広範囲に影響を与えないように
  障害発生時の迅速な対応を要求しているため

- Dockerのビルドでは
  パッケージング/ テスト/ クラウドレジストリへのプッシュまで
  のプロセスを完了させる必要があり、長時間かかる可能性がある

[7] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. 2014. Programmers' build errors: a case study (at google). In Proceedings of the 36th International Conference on Software Engineering (ICSE). ACM, 724–734

文献3 Docker におけるビルドの失敗に関する調査

RQ2: ビルドを修正するのにどれくらい時間がかかるか？

プロジェクトあたりのビルド失敗数
→ 修正時間の中央値と正の相関がある

失敗が多すぎると、開発者の時間と集中力が散漫になり、
困難な失敗や重要な失敗を時間内に修正できないことがある

[7] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. 2014. Programmers' build errors: a case study (at google). In Proceedings of the 36th International Conference on Software Engineering (ICSE). ACM, 724–734

文献3 Docker におけるビルドの失敗に関する調査

RQ3: 失敗の頻度と修正時間は時間の経過とともにどのように変化するか？

ビルド失敗率とその修正時間が時間の経過とともに増加する

６ヶ月間ごとのビルド失敗率(左)と修正時間(右)の変化

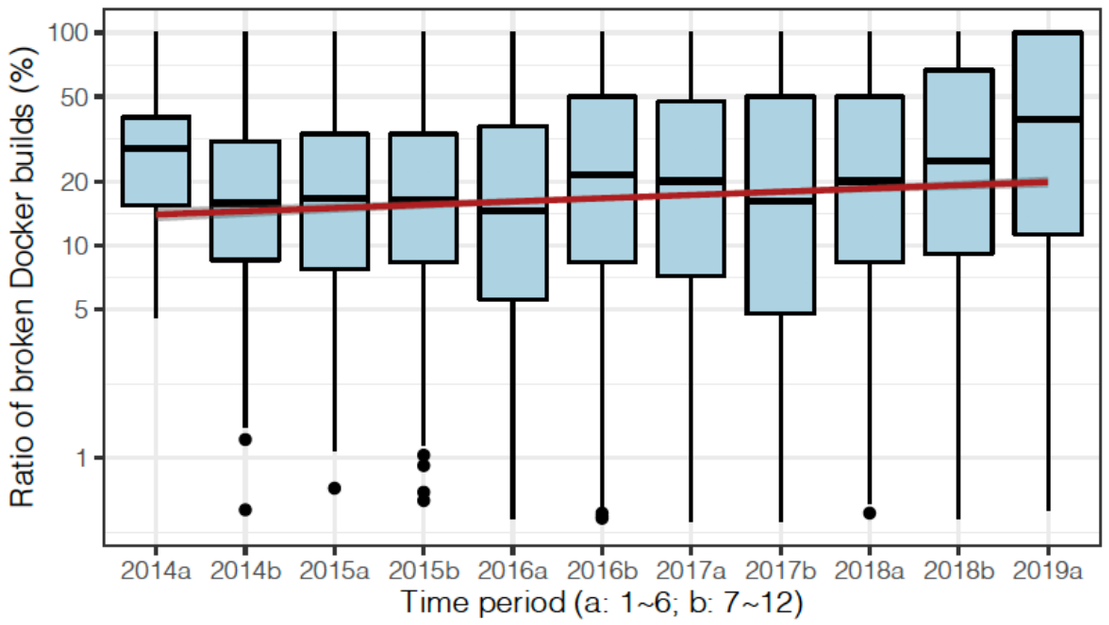


図. 6ヶ月ごとのDockerビルド失敗率の分布

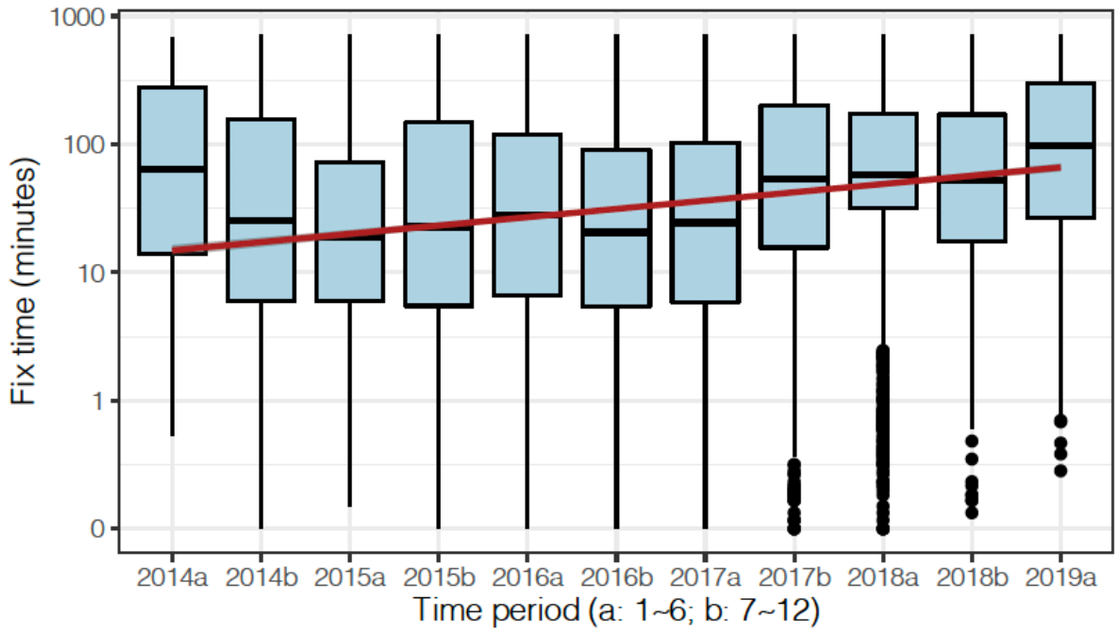


図. 6ヶ月ごとのDockerビルド修正時間の分布

文献3 Docker におけるビルドの失敗に関する調査

まとめ

- ビルドの修正時間は従来の研究より長く、
  ビルドの失敗数とその修正時間は時間の経過とともに長くなる

  Dockerにおける開発手法についての知見が不足している

→ **技術的な負債が含まれる可能性が高く、**
負債についての調査はビルド失敗時の修正を助ける手段になる

今後の展望

今後の展望

文献3で示されたように、Dcokerは比較的新しい技術であり
その開発手法が確立されていない

先行研究の手法を用いて
SATDについて調査を行うことで、
開発手法についての知見や**開発者をサポートするツール作成**に
つながる

今後の展望

DockerにおけるSATDの削除について

- 削除されている割合
- 削除のパターン・手法についての調査

調査を通して

- SATD 埋め込み時の推奨事項の提供、
  BOT への組み込みを行い、
  開発者を支援するツールを作成する
- Dockerでの調査を用いて既存研究との比較を行い、
  Dockerならではの特徴を分析する