Self-Admitted TechnicalDebt の削除 およびコンテナ仮想化技術の課題に関する調査

九州大学大学院　システム情報科学府
情報知能工学専攻　亀井研究室
修士 1 年　新堂 風

令和 2 年 12 月 14 日 (月) 16:40-17:25

1. はじめに

技術的負債とは，コード中に存在するバグや解消すべき課題のことであり，その中でも開発者が課題を把握したうえで，コードに埋め込まれた技術的負債を Self-Admitted Technical Debt(SATD) を呼ぶ．SATD を解消することは，ソフトウェアの品質向上につながることから，近年様々なアプローチで研究が実施されている．

他方，近年ソフトウェアのクラウド化に伴い，コンテナ型仮想化技術のひとつである，Docker が注目されている．Docker では，コンテナ型の仮想環境を作成，配布，実行することができ，軽量で高速に実行し，停止できるため，様々なプロジェクトで利用されている．Docker においても，従来の SATD 研究で調査対象とされてきた一般的なプログラミング言語と同様に，SATD の存在が報告されている [1]．しかし，Docker における SATD の削除についての調査はまだ行われていない．SATD の解消手法やそのパターンは，開発者を支援する知見としてとても有効であるため，調査するべき課題である．

そこで発表者はコンテナ型仮想化技術（Docker）における SATD の解消に関する調査を行っている．具体的には，SATD の解消期間や解消方法を調査し，Docker における SATD の解消パターンや特徴を，開発者を支援する知見として提供する予定である．

本発表では，Java のオープンソースプロジェクトにおける SATD の解消や，Docker におけるバグ修正に関する論文の調査結果を報告する．第 2 章では 5 つの大規模なオープンソースプロジェクトにおける SATD の解消期間や解消の割合についての調査結果，第 3 章では SATD 解消時のソースコードやコミットメッセージへの影響についての調査結果，第 4 章では Docker におけるビルドの失敗率とその修正期間についての調査結果を報告し，第 5 章でまとめと今後の展望について述べる．

2. SATD 削除期間や割合についての調査 [2]

2·1 概要

近年では，コードから SATD を検出する手法の開発や，ソフトウェア品質への影響を調査する研究が実施されている．しかし，SATD はプロジェクト内に 10 年以上存在する場合もあり，これは全ての SATD が「悪い」とされているわけではなく，削除すべき SATD もあれば，削除しなくても問題ない SATD があることも示している．そこで文献 [2] では，SATD の削除に焦点を当てた研究を行っている．5 つのオープンソースプロジェクトの Java ファイルについて，以下の 4 つの RQ に基づいて調査を行っている．

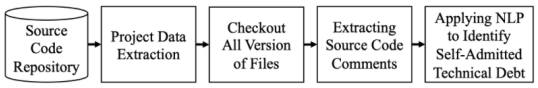

図 1: データ処理工程 (出典：文献 [2])

- RQ 1: どの程度の SATD が削除されているのか？
- RQ 2: 誰が SATD を削除しているのか？
- RQ 3: SATD はどれくらいの期間，プロジェクト内に存続するのか？
- RQ 4: SATD の削除はどのような活動によって行われるのか？

2·2 データの前処理

ここでは，文献 [2] で用いられてるデータとその事前処理について説明する．文献 [2] では，より多くの数・種類の SATD について，その変化を調査するために，コメント数が多く，活動性が高いプロジェクトに焦点を当てている．選定された 5 つのオープンソースプロジェクト（Camel、Gerrit、Hadoop、Log4j、Tomcat）は，Java を用いて Git 上で開発されている．

図 1 にデータ処理の工程を示す．Extractting Source Code Comment では，明らかに負債ではないライセンスコメントや IDE による自動生成コメントを削除している．Applying NLP（Natural Language Processing）では SATD のコメントを特定するために，Maldonado ら [5] の手法を用いている．NLP 分類器は [5] によって提供されたデータを用いて訓練されたものを使用している．

各工程後に取得したデータの詳細を表 1，及び，表 2 に示す．

表 1: 調査対象プロジェクトの詳細 (出典：文献 [2])

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

表 2: 調査対象プロジェクト内 SATD の詳細 (出典：文献 [2])

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

2·3 結果

従来の研究では，SATD の検出，管理，影響について検討されてきたが，そのような SATD の削除については，ほとんど知られていなかった．文献 [2] では，SATD の削除について調査を行なっており，その結果，SATD の大部分は削除され（プロジェクトごとの削除率，平均 74.4 %），また大部分は SATD を導入した本人により削除され（プロジェクトごとの自己削除率，平均 54.4 %），削除されるまでの期間は平均 82〜613.2 日であることがわかっている．さらに 14 名の開発者を対象に，SATD の導入と削除の理由について調査を実施している．調査の結果，開発者は SATD を削除する必要性を認識しているにもかかわらず，SATD を削除するための正式なプロセスはなく，ほとんどの削除はバグ修正の一環として行われていることを発見している．したがって，文献 [2] では，プロジェクトが効果的かつ体系的に SATD に対処できるような技術が必要であるということが示唆されている．

3. SATD 削除時のソースコードへの影響についての調査 [3]

3·1 概要

文献 [2] では，SATD の削除について主に定量的な観点をもとに調査を行っているが，削除の手法についての詳細な調査は行われていなかった．SATD の削除の手法の調査を行うことで，特定の種類の SATD に対応するためのパターン学習や，それに基づいて開発者への開発

表 3: データ詳細 (出典：文献 [3])

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

手法の提供を行うことが可能になる．文献 [3] では，[2] をもとに，5 つのオープンソースプロジェクトにおいて SATD がどのように削除されているかについての詳細を，以下の 3 つの RQ に基づいて調査している．

- RQ 1: SATD が削除された際の手法（クラス削除，メソッド削除など）はどのように分類されるか？
- RQ 2: SATD の削除はコミットメッセージに反映されているか？
- RQ 3: SATD を削除した際にコードにどのような変更が行われているか？

3·2 データの前処理

文献 [3] では，文献 [2] で用いられてるデータを利用し，調査を行っている．

表 3 にデータの詳細を示す．表 3 では，SATD の重複や不整合を除外したため文献 [2] より総数が少なくなっている．

また，SATD 削除の際のソースコードの変更に着目した調査を行うため，ファイルやクラス全体の SATD は無視して、メソッドレベルの SATD に着目している．その後，SATD の削除として不適切な例として，(1) SATD を含むファイル名の変更が削除と判定されている場合，(2) SATD がコード内の別の場所に移動されている場合，(3) SATD が構文的に変更されたが変更後も SATD を表している場合の 3 パターンを除外し，表 4 にその結果を示す．

次にコミットメッセージについて，コミットメッセージが SATD の削除について言及しているかどうか判断するために，コミットメッセージと削除された SATD

表 4: SATD 削除フィルタリング結果 (出典: [3])

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

とのテキストでの類似性をコサイン類似度を用いて分析し，類似している場合を手動で 5 段階に分類している．

3·3 結果

文献 [3] では，SATD の削除とソースコードの変更との関係をより深く調査している．その結果，SATD の削除はクラス全体またはメソッド全体が削除された場合に「偶発的（SATD の削除を目的とせず）」に発生していることが明らかになっている．さらに，SATD の削除がコミットメッセージに明言されているケースは，わずか 8% であり，SATD がどのように削除されるかについては，開発者は複雑な変更を行うだけでなく，メソッドの呼び出しや条件式を変更する傾向があるということが報告されている．

文献 [3] では，これらの知見を利用して，SATD 埋め込み時の推奨事項の提供や，自動アラートボットへの組み込みを通して，SATD を解決するための提案を提供することができると示唆されている．

Docker においても SATD の管理，解決のために，このような推奨事項の提供やボットの開発は有効であると考えられるため，調査を行うべきである．

4. Docker におけるビルドの失敗に関する調査 [4]

4·1 概要

Docker は近年，コンテナ型仮想化技術の世界標準として注目を浴びている．しかし，Docker のビルドは頻繁に失敗し，その修正には長い時間を要することが度々ある．これまでの研究では，大規模開発での Docker のビルド失敗率について調査した研究はあったが，失敗の頻度とその修正に関する調査を行った研究はほとんど存在していなかった．文献 [4] では，GitHub にリンクされた 3,828 件のオープンソースプロジェクトの 857,086 件の Docker のビルドについて調査している．Docker のビルドデータを用いて，以下の 3 つの RQ について調査を行っている．

- RQ 1: Docker のビルドはどれくらいの頻度で失敗するか？
- RQ 2: ビルドの修正を行うのにどれくらいの時間を要するか？

表 5: 各プロジェクトにおけるビルド数の基本統計 (出典：文献 [4])

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

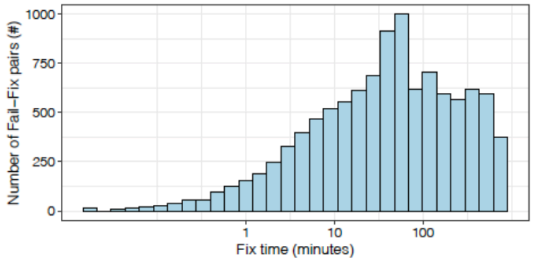

図 2: 修正時間の分布 (出典：文献 [4])

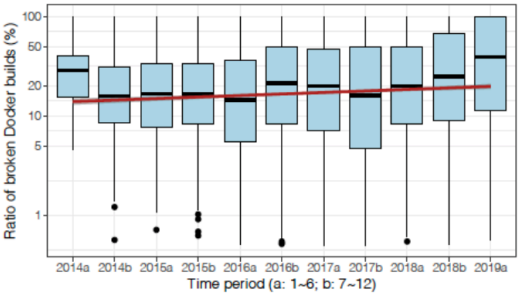

図 3: 6 ヶ月間のビルドの失敗率の分布 (出典：文献 [4])

- RQ 3: ビルド失敗の頻度と修正時間は時間の経過によってどのような関係性があるか？

4·2 データ収集

ここではデータの収集について説明する．文献 [4] では，公開されている Docker プロジェクトのビルドデータを取得するため，DockerHub API で “is_automated” という文字列の有無をチェックして公開されているプロジェクトの特定を行っている．特定したプロジェクトからビルド数が 10 以下のプロジェクトをフィルタリングして，最終的に 3,828 件のプロジェクトのデータセット，857,086 件の Docker ビルドデータ（メタデータを含む）を取得している．表 5 に各プロジェクトにおけるビルド数の統計値を示す．

4·3 データ処理

RQ2 についてビルドの修正時間にどれくらいの時間を要するかを分析するために，バグとその修正のペアの特定を行っている．

具体的には，ビルドが成功した直後に続く失敗したビルド (A) を発見し，次に発見する成功したビルド (B) の AB のペアを，ビルドの失敗とその修正としている．ここで開発者のスケジュールが修正時間に影響しないよ

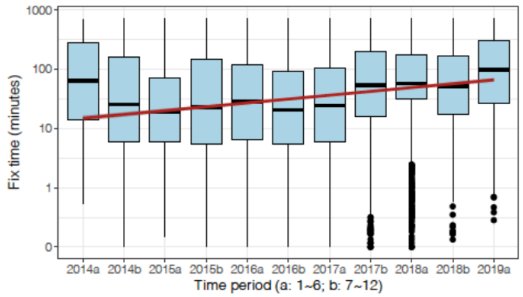

図 4: 6 ヶ月間の修正時間の分布 (出典：文献 [4])

うに，修正時間が 12 時間を超えるものをフィルタリングしている．図 2 に Docker ビルドの修正時間の分布を示す．

また，RQ3 について Docker のビルドの失敗とその修正時間の，時間経過による関係性を調査するため，6 ヶ月間の期間ごとにデータを集計する．その結果を図 3，図 4 に示す．

4·4　結果

文献 [4] では，Docker 環境におけるビルドの失敗事例について，大規模な実証実験を行っている．その結果，対象にした 3,828 件のプロジェクトにおいて，85.2% のプロジェクトでビルドが発生しており，31.5% のプロジェクトにおいて，ビルドの 20% 以上が失敗していることが明らかになっている．失敗したビルドの修正時間については，中央値が 44.2 分であり，ビルドの失敗が頻繁に発生するプロジェクトでは，修正時間が長いという結果も示されている．この結果は従来のビルドプロセスよりも長い時間になっている．文献 [4] では，Docker のビルドは他の言語と異なり，ビルド完了までのプロセスが長いために，修正に長い時間がかかりビルドの失敗が多いほど修正時間も長くなっていくと示唆している．また，図 3，及び，図 4 より Docker のビルドが失敗率やその修正時間は，時間の経過とともにどちらも増加する傾向があるということが示されている．

文献 [4] より，Docker ではビルドの失敗と修正に時間がかかることがわかり，その原因として Docker における開発手法の知見が不足していることが考えられる．また同時に，Dockerfile 内にビルドの失敗に関係するような SATD が含まれることが考えられ，SATD の返済について調査を行うことは，Docker におけるビルドの失敗や潜在するバグの問題を解決するのに有効であると考えられる．

5.　まとめ

第 2 章では 5 つの大規模なオープンソースプロジェクトにおける SATD の解消期間や解消の割合についての調査結果，第 3 章では SATD 解消時のソースコードやコミットメッセージへの影響についての調査結果，第 4 章では Docker におけるビルドの失敗率とその修正期間についての調査結果を報告した．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ことは，文献 [4] にて明確になっている．文献 [3]，文献 [3] のような SATD の削除に関する調査を行うことは，Docker におけるベストプラクティスな開発手法や SATD 削除に関する知見の提供，開発支援を行うツールの作成につながると考えられる．今後はこれらの調査手法や結果を参考に，Docker における SATD の解消に関する調査を行っていく．

参考文献

[1] 東英明, まつ本真佑, 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, pp.25-30, 2020

[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), pp.238-248, 2017

[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), pp.526-536, 2018

[4] Yiwen Wu, Yang Zhang, Tao Wang, and Huaimin Wang, “An Empirical Study of Build Failures in the Docker Context.” Proceedings of the 17th International Conference on Mining Software Repositories, pp.76–80, 2020

[5] E.d.S.Maldonado, E.Shihab, and N.Tsantalis, “Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt” IEEE Transactions on Software Engineering, pp.1044-1062, 2017

講演者　　：新堂　風
指導教員　：亀井 靖高 准教授
講演日時　：2020 年 12 月 14 日（月）16:40〜17:25
講演場所　：Teams