Vol.2021-SE-207 No.21
2021/3/1

Dockerfileにおける Self-Admitted Technical Debtの削除に関する調査

新堂 風[1,a)]　東 英明[2,b)]　柗本 真佑[2,c)]　亀井 靖高[1,d)]　鵜林 尚靖[1,e)]

概要：Self-Admitted Technical Debt (SATD) 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認識した上で，コードに埋め込んだものを指す．SATD の調査は，ソフトウェアの品質向上につながることから，SATD の追加や削除について様々な研究が行われている．他方，近年ソフトウェアのクラウド化に伴い，コンテナ仮想化技術の一つである Docker が注目されている．Docker は一般的な仮想環境と比べて，可搬性やリソース効率性が高く，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．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 解消実態の把握により，SATD 修正パターンの獲得や修正案の提示といった応用が期待できる．そこで本研究では，Docker Hub の人気上位 250 イメージを構築する Dockerfile を対象に，Dockerfile における SATD の削除の性質理解のための調査を行う．調査の結果， Dockerfile 内の SATD のうち，38.4%が削除されていた．また，削除された SATD のうち追加した本人により削除されていた割合は，68.8%であった．削除された SATD の存在期間は，中央値が 67 日であり平均値は 166 日であった．

キーワード：Self-Adimitted Technical Debt，技術的負債，コンテナ仮想化，Docker

1. はじめに

Self-Admitted Technical Debt(SATD) 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認識した上で， コードに埋め込まれたコメントを指す [1]．これまで，SATD に関する様々な研究が行われており，Sultan らによる SATD がソフトウェアの品質に与える影響についての研究 [2] では，SATD がソフトウェアに悪影響を及ぼす可能性があることが示唆されている．SATD の調査は，ソフトウェアの品質向上のための重要な課題であると考えられる．

他方，近年ソフトウェアのクラウド化に伴い，コンテナ仮想化技術の一つである Docker が注目されている．Docker は一般的な仮想環境と比べて，可搬性やリソース効率性が高く，様々なプロジェクトで利用されている．Docker ではテキスト形式で環境が構築できるため，一般的なプログラミング言語と同様にコメントが存在し，SATD の存在も報告されている [3].

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．これまで，Java を対象とした SATD 削除の実態調査が Maldonado ら [4] や Zampetti ら [5] によって行われており，調査結果は既存の SATD 解消やソフトウェアの品質向上のための知見となっている．Docker においても同様に，SATD の削除手法やそのパターンは開発者を支援する知見として有用であるため，調査するべき課題である．

本研究では，Dockerfile における SATD 削除の実態把握を目的としている．目的を達成するため，以下 3 つの Research Question（RQ）を設定し，Docker Hub の人気上位 250 イメージを構築する Dockerfile を対象に調査を行なった．

RQ1: どの程度の SATD が削除されているか？

対象の Dockerfile から 125 件の SATD を検出し，そのうち 48 件（38.4%）が削除されていた．

[1] 九州大学大学院システム情報科学研究院
Kyushu University
[2] 大阪大学大学院情報科学研究科
Osaka University
[a)] shindo@posl.ait.kyushu-u.ac.jp
[b)] h-azuma@ist.osaka-u.ac.jp
[c)] shinsuke@ist.osaka-u.ac.jp
[d)] kamei@ait.kyushu-u.ac.jp
[e)] ubayashi@ait.kyushu-u.ac.jp

RQ2: SATD を追加した開発者が削除を行っている割合はどの程度か？

削除されている 48 件の SATD のうち，同一開発者による自己削除が行われているのは 33 件（68.8%）であった．

RQ3: SATD の存在期間はどの程度か？

削除された SATD の追加日時と削除日時の差を存在期間とし，調査を行なった結果，存在期間の中央値は 67 日であり平均値は 166 日であった．

以降，第 2 章では本研究に関連する研究，第 3 章では本調査手法について説明する．第 4 章，第 5 章では調査結果についての説明とその考察を行う．第 6 章では，妥当性への脅威について述べ，第 7 章で結論と今後の課題について述べる．

2. 関連研究

2.1 技術的負債

技術的負債とは，コードに存在するバグや解消すべき課題のことであり，ソフトウェアの品質向上のために解決されるべきであると考えられている．また，技術的負債という概念を導入した Cunningham[6] は，技術的負債の悪影響は開発者の開発力向上によって，過去自身が作成した品質の悪いシステムの解読が難化することによる生産性低下のことであり，リファクタリングなどにより解消すべきであるという見解も示している．技術的負債の中でも，特に開発者によって意図的に混入された技術的負債を Self-Admitted Technical Debt（SATD）と呼ぶ．

Wehaibi ら [2] は，SATD がソフトウェアの品質に与える影響に関する分析結果を報告している．調査は Java，C，C++，Scala，及び JavaScript の 5 つのオープンソースプロジェクトを対象にしている．Sultan らは，SATD が含まれるファイルと含まれていないファイルに存在する欠陥を比較し，SATD とソフトウェアの品質の関係を調査している．また，SATD を含むコードの変更はそれが含まれないコードの変更よりも，複雑で困難であるかどうかを調査している．その結果，SATD と欠陥については明確な関係性はないものの，SATD の変更は複雑で困難なものが多いため，将来的にシステムを変更することを困難にしているという点で悪影響があることを示している．そのため，Sultan らは SATD を適切に管理する必要があると示唆している．

Maldonado ら [4] は，SATD には 10 年以上プロジェクトに存在するものがあることから，削除すべき SATD とそうでない SATD の存在に着目し．SATD 削除の実態調査を行った．Java を対象に調査が行われ，結果として SATD の削除割合や削除するまでの期間，また削除する人物についての知見を明らかにした．さらに，Zampetti ら [5] は，Maldonado らの研究をもとに SATD の削除手法についてより詳細な調査を行い，Java における SATD 削除手法のパターンを明らかにしている．これらの知見は，開発者が SATD を対処する際に有用であり，将来的に ReviewBot[7] のような自動コード再表示ボットに組み込むことができると考えられている．

これらの SATD がソフトウェアの品質に与える影響の研究や SATD の削除についての研究を含め，SATD に関する多くの研究が行われており，その中で SATD はソフトウェアの品質を向上させるための解消すべき課題の一つであると述べられている．

2.2 Docker

コンテナ仮想化技術とは，アプリの実行環境を構築する一種の仮想化技術のことである．コンテナ仮想化技術の一つである Docker は可搬性やリリース効率性の高さから，様々なプロジェクトで利用されている．Docker は比較的新しい技術であり開発の知見が少ないため，一般的なプログラミング言語と比較して開発上の課題が多いことが考えられる．本節では，Docker 開発上の課題について関連研究を示し，Docker における SATD について調査することの重要性を示す．

Wu ら [8] は，Dockerfile におけるコードの臭いの存在について，その発生頻度やプロジェクトの特性との相関関係を把握することを目的とした実証的調査を行った．6,334 件の GitHub プロジェクトを対象にした調査により，約 83.8%のプロジェクトの Dockerfile 中に少なくとも 1 つの臭いが存在することを明らかにした．特に，Dockerfile のベストプラクティス[*1] に違反することによる臭いが多いと示している．ベストプラクティスに違反する原因の一つとして，Docker における開発者の知識や能力が不足していることが考えられる．

さらに，Wu ら [9] は Docker におけるビルドの失敗率とその修正に要する期間について研究を行った．その結果，Docker では失敗したビルドの修正に要する時間は，他のプログラミング言語と比較して長く，また，ビルドの失敗率とその修正時間は，プロジェクトが進むにつれてともに増加することが明らかになっている．Yiwen らは，Docker における開発手法についての知見が不足し

[*1] https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

ており，修正が困難なバグが多いことを示している．

Docker にも他のプログラミング言語と同様に SATD の存在が確認されており，開発知見の少ない Docker における SATD の性質は，開発者にとって有益な知見となるため明らかにすべきであると考えられる．

2.3 Docker における SATD

既存の SATD 研究では，調査題材として Java が広く用いられており，Docker を題材にしている研究はほとんど存在しない．Docker では，作成したイメージが多くの開発者に広く利用されることから，他のプログラミング言語に比べて SATD が波及に繋がりやすい可能性がある．そのため，Docker においても SATD の性質やその削除手法などの実態調査は，SATD の波及を抑制するために重要となる．

東ら [3] は，Docker における SATD の存在やその分類について研究を行っており，Docker にも一般的なプログラミング言語と同様に SATD が含まれることを明らかにしている．調査の結果，Dcoker コメント内のコメントの内，約 3.4%が SATD に関する記述であり，さらにバージョン固定に関する SATD や PGP 等を用いた真正確認に関する SATD など，コンテナ仮想化技術固有の SATD を発見している．

調査課題：本研究では Docker に含まれる SATD の削除についての調査を行う．はじめに，本研究は Docker における SATD の削除の性質を明らかにする基礎となる研究であり，Docker における SATD がどの程度削除されているかを調査する（RQ1）．また，Maldonado ら [4] による Java における SATD の削除に関する調査と同様の調査を行い，結果を比較することで Docker 固有の特徴について知見を得ることができる．さらに，Docker ではイメージを共有し配布するため，SATD の存在期間はイメージの利用者にとって重要である．そのため，削除された SATD がそれを追加した開発者による自己削除であるかを調査し（RQ2），削除された SATD の存在期間を調査する（RQ3）．

3. 調査手法

本章では，データセットの構築から RQ 分析までの本研究の調査手法を説明する．図 1 に調査手法の流れを示す．以降では本図の流れに沿って調査手法を説明する．

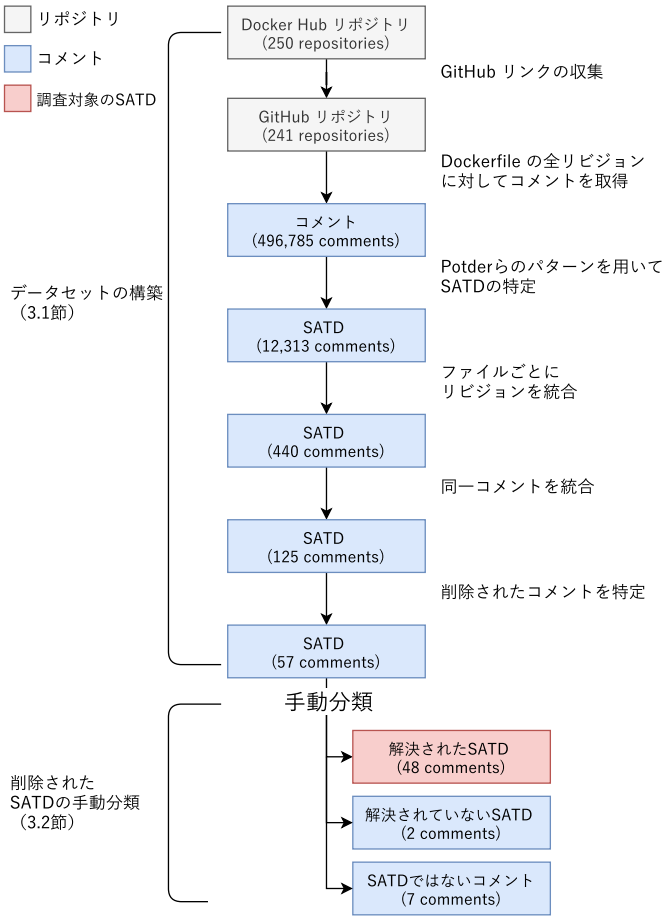

図 1 調査手法の概要

3.1 データセットの構築

3.1.1 対象プロジェクトの選定

本研究では，Dockerfile の過去から現在までの全リビジョンにおけるコメントを調査の対象とする．SATD は継続的に開発やメンテナンスが行われているプロジェクトに多く存在すると考えられる．また，Docker では作成したイメージが配布・再利用されるため，より広く利用されるプロジェクトに含まれる SATD を調査することは効果的であると考えられる．そこで本研究では文献 [3] と同様に，Docker の公式レジストリである，Docker Hub の人気リポジトリの Dockerfile からデータセットを構築する．

本研究では Docker Hub の人気上位 250 リポジトリを対象として，そのプロジェクトに紐づけられた GitHub リポジトリへのリンクを収集した結果．241 個のリンクを収集できた．さらにそのリンク先のリポジトリ内の Dockerfile を取得し，過去からデータ取得時（2020 年 09 月）までの全リビジョンのコメントを取得した結果，496,785 件のコメントを得られた．

3.1.2 SATD コメント検出

3.1.1 節で取得したコメントには，SATD ではないコメントも多く含まれる．そのため文献 [3] と同様に，Potder らが発見した “hack” や “fixme” などの SATD に含まれやすい 63 のキーワードのパターン [1] に，“todo” という文字列を追加した 64 のキーワードのパターンを含まないコメントを除去した．文献 [3] より，64 のキーワードによる SATD 検出の適合率は 86.4%であり，再現率は 57.6%，F 値は 0.69 と算出されており，キーワードを使用することで高い精度で SATD を検出することが可能であると示されている．キーワードによる SATD 検出の結果，12,313 件のコメントを得られた．

3.1.3 リビジョンの統合

3.1.2 節で取得したコメントは，各 Dockerfile の全リビジョンを対象にしているため，多くのコメントが重複して計測されている．そのためリビジョンを統合する必要がある．Zampetti ら [5] と同様に git を利用してファイル名の変更や移動を追跡し，リビジョンの統合を行う．また RQ2・RQ3 にて，SATD の追加・削除者と削除されるまでの存在期間を調査するため，リビジョンの統合時に git のコミットデータから（1）SATD を追加したコミットの日時，（2）追加時のコミット者，（3）SATD を削除したコミットの日時，（4）削除時のコミット者の情報を取得する．リビジョンの統合によりコメントは 440 件に集約できた．

3.1.4 コメントの統合

Dockerfile を扱う多くの GitHub リポジトリでは，OS やソフトウェアのバージョンにあわせた環境を作成している．そのため，同一リポジトリに数行のみの違いがある Dockerfile が存在していた．これらの類似した Dockerfile には，それぞれ同じ SATD が存在していることが多く，SATD の削除に関する本研究においてはバイアスをうむ可能性が高いと考えられる．また，SATD が複数の Dockerfile に存在している場合でも，それらの追加・削除日は一致していた．そこで，それらの SATD を同一として扱うため，単一のコメントに統合した．その結果，コメント数を 125 件に集約できた．

3.1.5 削除されたコメントの抽出

ここでは検出した 125 件のコメントから，削除されているコメントを抽出する．削除にはファイルごとコメントが削除されている場合と，ファイルは削除されずにコメントが削除されている場合の 2 通りが存在しているが，本研究では SATD の削除手法から，SATD を削除する際に役立つ知見を得ることを目的としているため，ファイルは削除されずにコメントが削除されている場合に絞って調査を行う．ファイルごと削除されているコメントを除外した結果，57 件のコメントが得られた．

表 1　SATD 削除についての分類結果

分類定義	件数
負債が解決されている	48 / 57 (84.2%)
負債が解決されていない	2 / 57 (3.5%)
SATD ではない	7 / 57 (12.3%)

3.2 SATD が解決されているか手動分類

3.1 節では，Dockerfile の全リビジョンから削除されている SATD を抽出しているが，負債を解決していないまま SATD を削除しているケースでは，SATD の削除に役立つような知見を得ることはできないと考えられる．そのため，SATD の削除の手法について手動分類を行うことで，本研究の対象ではない負債が明らかに解決されていない SATD を除外する．本研究の手動分類は第 1，第 2，第 3，第 4 著者の 4 名の著者によって行い，削除手法の分類と議論を 3 回に分けて行った．

3.2.1 Phase1: 10 件の分類

はじめに 57 件の SATD のうち 10 件の SATD をランダムに抽出して分類を行なった．著者 4 名がそれぞれ 10 件の SATD について，追加時・削除時のコードの変化やコミットメッセージなどから SATD がどのように削除されたかを文章形式で表した．続いて，各々の調査をもとに SATD の削除について分類カテゴリを議論し，分類カテゴリと判断基準を作成した．

3.2.2 Phase2: 25 件の分類

Phase1 で作成した分類カテゴリをもとに，25 件の SATD について著者 4 名それぞれが手動分類を行い，その後分類カテゴリの議論を行なった．また，同時に分類カテゴリとその基準について再調整を行い，Phase1 で分類した 10 件をあわせて新カテゴリで再分類した．

3.2.3 Phase3: 22 件の分類

Phase3 では，残りの 22 件の SATD について Pahse2 と同様の方法で分類と議論を行なった．

表 1 に分類結果を示し，それぞれの分類の代表例をスニペット 1[*2]，スニペット 2[*3]，スニペット 3[*4] に示す．

スニペット 1 の負債解決の例では，コメント上部のコードに “linux/386・linux/arm” を加えることを記載してお

[*2] https://github.com/docker/docker-ce/commit/df82456ed97e08ffa192c1f821acd9315295e7da
[*3] https://github.com/docker/docker-ce/commit/503a6ed4ffd6af08d03275cdb5989d818afcbcff
[*4] https://github.com/docker/docker-ce/commit/2652782fc983bd35c9e2a7a888c3ebb595cc31e1

スニペット 1　負債解決の例

```
1 - ENV DOCKER_CROSSPLATFORMS darwin/amd64 darwin/386
2 - # TODO add linux/386 and linux/arm
3 + ENV DOCKER_CROSSPLATFORMS linux/386 linux/arm darwin/amd64
        darwin/386
```

スニペット 2　負債が解決されていない例

```
1 - # install seccomp
2 - # TODO: switch to libseccomp-dev since dockerinit is gone
3 + # install seccomp: the version shipped in trusty is too old
```

スニペット 3　SATD ではないコメントの例

```
1 - # Options for hack/validate/gometalinter
2 - ENV GOMETALINTER_OPTS="--deadline=2m"
```

り，削除時にはコメント通りに “linux/386・linux/arm” を加えている．そのため負債が解決されていると判断している．スニペット 2 の負債が解決されていない例では，TODO というコメントは削除され，64 のキーワードパターンを含まないコメントになっているが，“the version shipped in trusty is too old” という文脈から依然として負債であると考えられる．またスニペット 3 の SATD ではないコメントの例では，コメント内に “hack” というキーワードは存在しているが，単にパスに含まれていた単語であるため，SATD ではないと判断している．

4. 調査結果

本章では，3 章で取得した 48 件の削除された SATD について，3 つの RQ の調査結果を報告する．

4.1 RQ1: どの程度の SATD が削除されているか？

目的： 一つ目の RQ は SATD の削除の割合についてである．過去の研究で，技術的負債の発生は避けられないものであり，ソフトウェアに悪影響を及ぼすことが述べられている [10]．SATD についても多くの研究が実施され，同様にソフトウェアに悪影響を及ぼすことが明らかにされている [2]．Docker において，開発者がどのように SATD を対処してきたか知るために，まずはどの程度の SATD が削除されているか調査を行う．

アプローチ： 3 章にて，Docker Hub の人気上位 250 リポジトリを対象として，Dockerfile を取得しコメントを抽出した．また，東ら [3] と同様に SATD に含まれやすい 64 のキーワードのパターンを使用して，コメントから SATD を特定した．さらに，全リビジョンを遡ってコメントを取得し，SATD が存在している最後のリビジョンより後に変更が行われている場合は，その SATD が削除されたと判断した．

結果： Docker Hub の人気上位 250 リポジトリを対象に，Dockerfile に含まれる 125 件の SATD を取得した．そのうち削除されていたのは 48 件（38.4%）であった．

> 特定した 125 件の SATD のうち，38.4%が削除されていた．

4.2 RQ2: SATD を追加した開発者が削除を行っている割合はどの程度か？

目的： 二つ目の RQ は SATD の削除を行う開発者についてである．Maldonado ら [4] による研究では，Java において SATD は追加者自身によって削除される割合が高いことが明らかにされている．これは追加者自身の方が SATD への理解が深く，対処しやすいからであると推測できる．Docker は比較的新しい技術であり開発の知見が少ないため，追加者自身も SATD の理解が浅い可能性がある．本 RQ の調査により，Docker の SATD 削除についての性質を人物の観点から明らかにする．

アプローチ： 削除されている SATD について，追加時と削除時のコミット情報から変更を行った人物の情報を取得した．人物の情報は，GitHub のアカウント名とメールアドレスから構成されており，これらの情報はいつでも変更可能である．削除されている 48 件について，目視確認したところ開発中にメールアドレスのみ変更されている例が確認できたため，追加時と削除時のアカウント名が一致しているかどうかを判断した．

結果： 削除されている SATD の 48 件のうち，自己削除されている SATD の件数は，33 件（68.8%）であった．しかし，対象の Dockerfile の開発人数が 1 人であった場合は，必然的に自己削除になるため，考察を行うにはそれぞれの Dockerfile の開発人数を知る必要がある．git のコミットデータから，48 件の SATD についてそれぞれが削除されるまでの Dockerfile の開発人数を取得した．図 2 にその結果を示す．また，自己削除されている 33 件の分布を同図に橙色で示す．

全 48 件のうち，35 件（72.9%）は 5 人以下での開発が行われており，それらのうち 29 件（82.9%）で同一開発者による SATD の自己削除が行われていた．また，20 人以上が開発を行っている Dockerfile についても，全 7 件中 3 件（42.9%）で SATD の自己削除が行われていた．

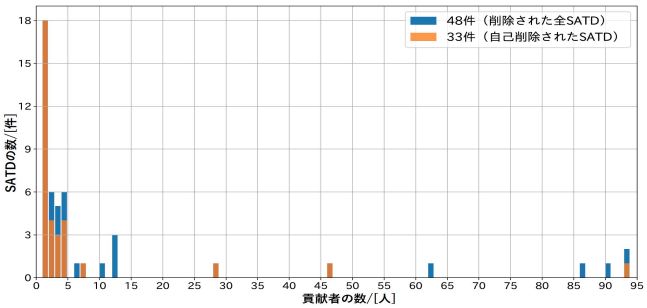

図 2　対象 Dockerfile の貢献者の数

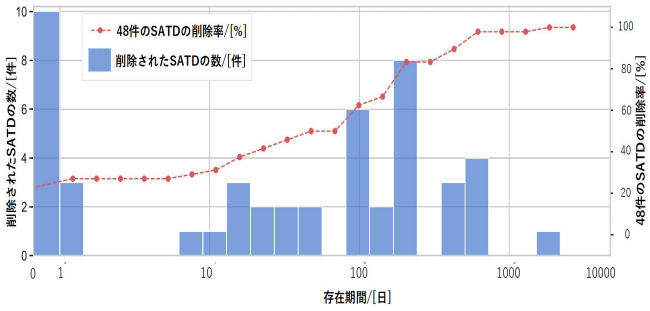

図 3　削除された SATD の存在期間 [日]

68.8%の SATD が追加者自身によって削除されていた．

4.3　RQ3: SATD の存在期間はどの程度か？

目的： 三つ目の RQ は SATD の存在期間についてである．Docker では既に作成されているイメージを親イメージとして使用することが多く，親イメージに存在する SATD は自身のプロジェクトにも影響するため，開発者は注意を払う必要がある．Docker における SATD の一般的な存在期間を知ることは，自身のプロジェクトに影響する SATD がいつまで存在するかを推測するための一つの指標になると考えられる．

アプローチ： 削除されている SATD を対象にし，SATD の追加時と削除時の日付情報を取得した．日付は分単位まで取得でき，追加時と削除時の日付の差を SATD の存在期間とした．また，存在期間の単位が “日” となるように計算を行った．

結果： 削除されている 48 件の SATD について，存在期間を日付単位で算出したグラフを図 3 に示す．図中の青色の棒グラフは，存在期間（日）について作成した度数分布図であり，赤色の折れ線グラフは 48 件に占めるそれまでに削除された SATD の割合である．存在期間の中央値は 67 日であり，平均値は 166 日であった．また図 3 より，10 件（20.8%）が 1 日未満で削除されていることがわかる．

また，スニペット 4[*5] に本研究で特定した SATD のうち削除までの期間が長い例を示す．この SATD の存在期間は 363 日であり，中央値の 67 日より長かった．示した例では，外部の問題が解決されたら親イメージを切り替えることをコメントに記述している．実際に，コメントにて言及されている問題が外部で解決された後に SATD を削除しているため，負債の原因が明らかに外部

スニペット 4　削除までの期間が長い SATD の例

```
1 - FROM php:7.2-apache-stretch
2 - # TODO switch to buster once https://github.com/docker-
        library/php/issues/865 is resolved in a clean way (
        either in the PHP image or in PHP itself)
3 + FROM php:7.4-apache-buster
```

スニペット 5　削除までの期間が短い SATD の例

```
1 - # TODO: Switch to debian sid
2 - FROM php:5.6-apache
3 + FROM debian:sid
```

プロジェクトに存在していると考えられる．負債の原因が外部に存在している場合は，SATD が削除されるまでの期間は外部に依存する可能性が高く，存在期間が長くなることが考えられる．

一方，スニペット 5[*6] に削除までの期間が短い SATD の例を示す．SATD の存在期間は 2 時間 47 分とかなり短く，コメントでは親イメージを “debian:sid” に変更することを記載していた．スニペット 5 のように，負債への対応方法が明確に記載されている場合は，追加してから削除するまでの期間が短くなる可能性が高い．このように SATD 自体の特徴と削除の性質は依存関係にある可能性があり，Docker における SATD の削除の性質を調査する上で重要であるため，今後調査を行う必要がある．

自己削除との関係性： 自己削除された SATD とそうでない SATD に分けて存在期間の調査を行った図を，図 4，図 5 に示す．自己削除された SATD の存在期間の中央値は 24 日であり，平均値は 158 日であった．また，自己削除されていない SATD については，中央値 110 日であり，平均値は 183 日であった．これらの結果より，自己削除された SATD はより早く削除されることがわかった．Maldonado ら [4] も同様の結果を示しており，Docker においても他プログラミング言語と同様の傾向があるといえる．Maldonado らは，SATD の追加者の方

*5　https://github.com/docker-library/drupal/commit/71ffa1cde017936514d24d64127fb78956ea5c1b

*6　https://github.com/wikimedia/mediawiki-docker/commit/1828584c75e34ee08f12427e64d907f469cbab7c

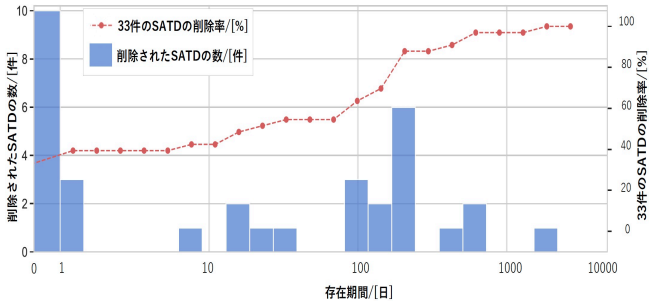

図 4　自己削除された SATD の存在期間 [日]

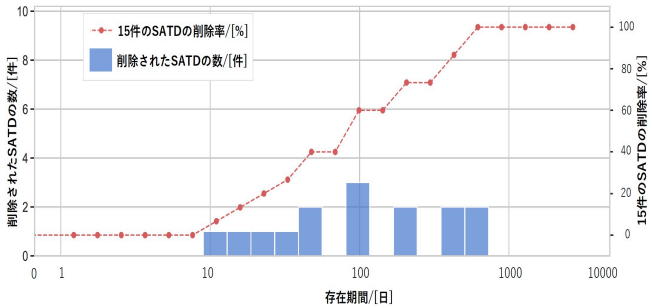

図 5　自己削除されていない SATD の存在期間 [日]

が他の開発者よりも SATD を対処しやすいのではないかと示唆しているが，Docker においてはより深い知識を持って開発が行える開発者が少ないことも原因として考えられ，定性調査などさらなる調査を行う必要がある．

SATD が追加されてから削除されるまで，中央値で 67 日，平均値で 166 日の期間があった．追加者自身により削除された SATD は，中央値で 24 日，平均値で 158 日の期間があった．

5.　考察

本稿では，Docker における SATD の削除についての定量的調査を行った．本章では，Docker における SATD ならではの特徴を明らかにするため，Java プロジェクトを対象に行われた Maldonado らの研究 [4] との比較を行い，調査結果について考察する．さらに，その結果が開発者，研究者のそれぞれに対してどのように役立つかを説明する．

5.1　従来研究との比較

本研究での Docker における SATD の削除についての結果，および Maldonado らによる Java を対象にした研究結果を表 2 に示す．表 2 より，Docker における SATD は，Java と比較して削除された SATD の割合が 36.0% 低く，自己削除されている割合は 14.4%高いことが明らかになった．それにより，Docker における SATD は削除が困難なものが多く，特定の開発者に頼った開発が行われていると考えられる．

表 2　SATD の削除についての調査結果の比較

調査項目	対象言語 Docker	対象言語 Java
削除されている割合	38.4%	74.4%
自己削除されている割合	68.8%	54.4%
存在期間の中央値	67.0 日	18.2〜172.8 日

5.2　開発者と研究者への貢献

5.2.1　開発者への貢献

Storey ら [11] は，ソフトウェア開発における TODO コメントなどの注釈の役割について研究しており，詳細が書かれておらず他人が理解できない TODO コメントは，メンテナンスに悪影響を及ぼすことを述べている．Docker における SATD が，Java と比較して削除率が低く，自己削除の割合が高い原因として，負債の詳細が書かれていないことが考えられる．

特に，Docker においては外部に原因がある SATD も存在し，本研究で外部に原因がある SATD は削除するまでの期間が長い可能性を示した．このような SATD については，外部の原因となる箇所を参照するリンクを必ず挿入するなどの対策をすることで削除するまでの期間短縮や，追加した本人以外の開発者による削除を可能にすることができると考えられる．本研究結果に加えてより詳細な調査を行うことで，開発者にとって有益な，Docker における SATD 追加時のガイドラインを整備することができる．

5.2.2　研究者への貢献

研究者への貢献として，Docker における SATD についての調査の重要性や，解決すべき課題について示す．

Docker においては外部の問題が自プロジェクト内で SATD として残されることがあり，開発者は常に外部の問題が解決されたかどうか確認しなくてはならない．このような課題を解決する方法として，外部の問題が解決されたら自動で開発者に通知を行い，SATD の削除を促すようなツールの作成が挙げられる．

また，5.2.1 節で述べたように，Docker における SATD は自己削除の割合が高いことから，SATD の詳細や解決方法を他人が理解できていない可能性が考えられる．Docker では一般的なプログラミング言語に比べて負債が伝搬しやすい可能性があるため，解決困難な SATD を生むことは，長期に渡り様々なプロジェクトに対して悪影響を及ぼす可能性が高いと考えられる．そのため，

SATD 追加時に詳細が不足している場合は，開発者にアラートを出すようなツールの作成や，Docker 開発における SATD 追加のフォーマットの作成を通して，今後解決しやすい SATD の作成を促す必要がある．

6. 妥当性への脅威

6.1 内的妥当性

本研究では，GitHub リポジトリ内に存在する Dockerfile を対象にした．これは，Docker の公式サイトにおいて環境構築ファイルに “Dockerfile” という名前が利用されており，Docker を利用する多くの開発者もこれに準じて “Dockerfile” を利用するからである．しかし，ファイル名に規則はないため，“Dockerfile_windows” などの名前を使用することが可能である．今回のように，同一の GitHub リポジトリ内で OS やソフトウェアのバージョンにあわせた環境を作成している場合は，環境ごとにファイル名を変更している可能性がある．そのため，対象にすべきファイルを取得できていない可能性がある．

6.2 外的妥当性

本研究では，対象にしたプロジェクトの数が外的妥当性への脅威となりうる．今回は，Docker Hub の人気上位 250 リポジトリを対象にしているが，調査結果の一般化のためには，より多くの SATD を含むプロジェクトを対象に，同様の調査を行う必要がある．

7. おわりに

本研究では，Docker における SATD の削除についての実態調査として，Dockerfile に存在する SATD が削除されている割合，また削除されるまでの期間と削除した人物について分析を行った．その結果，Dockerfille に含まれる SATD の 38.4%が削除されており，その 68.8%が自己削除されていることを明らかにした．SATD が削除されるまでの期間については，中央値が 67 日で平均値が 166 日であった．これらの結果を Java での研究結果 [4] と比較して，Docker における SATD は削除が困難なものが多く，特定の開発者に頼った開発が行われている可能性を示した．また，本研究結果より Docker における SATD の性質を調査することの重要性を明らかにした．

今後の展望としては，東ら [3] のカテゴリに従った SATD 自体の分類を行い，Docker に含まれる SATD の種類と合わせた，削除の特徴を調査する研究が挙げられる．また，3.2 節で行った SATD 削除の分類は，負債が解決済みか否かという点に着目しており，その削除手法の分類は行っていない．より詳細な削除手法の分類は，Docker における SATD の削除に関するさらなる知見につながるため，重要な課題であると考えられる．

謝辞

本研究の一部は JSPS 科研費 JP18H04097・JP18H03222，および，JSPS・国際共同研究事業の助成を受けた．

参考文献

[1] A. Potdar and E. Shihab. An exploratory study on self-admitted technical debt. In *In Proceedings of International Conference on Software Maintenance and Evolution*, pp. 91–100, 2014.

[2] S. Wehaibi, E. Shihab, and L. Guerrouj. Examining the impact of self-admitted technical debt on software quality. In *In Proceedings of International Conference on Software Analysis, Evolution, and Reengineering*, pp. 179–188, 2016.

[3] 東英明, 柗本真佑, 亀井靖高, 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, Vol. 120, No. 193, pp. 25–30, 2020.

[4] E. D. S. Maldonado, R. Abdalkareem, E. Shihab, and A. Serebrenik. An empirical study on the removal of self-admitted technical debt. In *In Proceedings of International Conference on Software Maintenance and Evolution*, pp. 238–248, 2017.

[5] F. Zampetti, A. Serebrenik, and M. D. Penta. Was self-admitted technical debt removal a real removal?: An in-depth perspective. In *In Proceedings of International Conference on Mining Software Repositories*, pp. 526–536, 2018.

[6] W. Cunningham. The wycash portfolio management system. *SIGPLAN OOPS Mess.*, Vol. 4, No. 2, p. 29–30, 1992.

[7] V. Balachandran. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *In Proceedings of International Conference on Software Engineering*, pp. 931–940, 2013.

[8] Y. Wu, Y. Zhang, T. Wang, and H. Wang. Characterizing the occurrence of dockerfile smells in open-source software: An empirical study. *IEEE Access*, Vol. 8, pp. 34127–34139, 2020.

[9] Y. Wu, Y. Zhang, T. Wang, and H. Wang. An empirical study of build failures in the docker context. In *In Proceedings of International Conference on Mining Software Repositories*, p. 76–80, 2020.

[10] E. Lim, N. Taksande, and C. Seaman. A balancing act: What software practitioners have to say about technical debt. *IEEE Software*, Vol. 29, No. 6, pp. 22–27, 2012.

[11] MA. Storey, J. Ryall, R. I. Bull, D. Myers, and J. Singer. Todo or to bug: Exploring how task annotations play a role in the work practices of software developers. In *In Proceedings of International Conference on Software Engineering*, p. 251–260, 2008.