# コンテナ仮想化技術における SATDの削除に関する調査

新堂 風[1,a)]　近藤 将成[1,b)]　柏 祐太郎[1,c)]　東 英明[2,d)]
柗本 真佑[2,e)]　亀井 靖高[1,f)]　鵜林 尚靖[1,g)]

**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 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 イメージを対象に，Dockerfile に含まれる 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 40.7%の負債が解決されており，存在期間は中央値が 67 日，平均値が 166 日であった．また，Dockerfile 自体のレビューを求める SATD や，外部システムに起因する SATD が多いことを明らかにした．外部システムに起因する SATD の早期解決を開発者に対して促すため，外部システムの変更を検知し，更新時に開発者に通知を行うツールを作成した．

**キーワード**：Self-Admitted 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．

[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)] kondo@ait.kyushu-u.ac.jp
[c)] kashiwa@ait.kyushu-u.ac.jp
[d)] h-azuma@ist.osaka-u.ac.jp
[e)] shinsuke@ist.osaka-u.ac.jp
[f)] kamei@ait.kyushu-u.ac.jp
[g)] ubayashi@ait.kyushu-u.ac.jp

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 はコードを書いて開発するという点で Java と類似しており，Docker においても，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．また，開発者による負債の解決を

支援するために有益な知見を獲得するため，削除されている SATD の中でも，特に負債が解決されているものについて調査を行う．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 が解決されているか？**

対象の Dockerfile から 118 件の SATD を検出し，そのうち 48 件（40.7%）の負債が解決されていた．

**RQ2: SATD の存在期間はどの程度か？**

解決された SATD の追加日時と解決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．

**RQ3: どのような種類の SATD が解決されているか？**

Dockerfile 自体のレビューを求める SATD や，外部システムに起因する SATD が解決されていた．

これらの発見をもとに，本研究では，開発者に対して SATD の早期解決を促すための支援ツールを作成し，GitHub 上で公開[*1] した．

以降，2 章では本研究に関連する研究，3 章では本調査手法について説明する．4 章，5 章では調査結果についての説明とその考察を行う．6 章では，作成した開発者支援ツールについて述べる．7 章では，妥当性への脅威について述べ，8 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とは，過去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，開発における生産性低下のことであると述べている．そのため，Cunningham[6] は，リファクタリングなどにより技術的負債を解消すべきであるという見解を示している．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 [2] 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 [2] 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 [4] 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，開発上の課題が多いことが考えられる．

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[*2] 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

[*1] https://github.com/posl/notify-when-versionup-action

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

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期間は，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期間は，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．Wu ら [9] は，Docker における開発手法についての知見が不足しており，修正が困難なバグが多いことを示している．

### 2.3 Docker における SATD

Docker においても，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いる．しかし，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，任意のイメージを継承できる機能が存在し，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，一般的なプログラミング言語に比べて SATD が他プロジェクトへ波及し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．

Azuma[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，バージョン固定に関する SATD や Pretty Good Privacy（PGP）等を用いた真正確認に関する SATD などが発見されている．Docker では外部のファイルやソフトウェアの利用が必要になる場合が多いため，このような SATD が確認できたと考えられており，Azuma[3] はこれらの SATD が Docker 特有のものであると主張している．また，Docker では一般的なプログラミング言語と異なり，イメージを構築する手続き命令の集合を記述する．そのため，一般的なプログラミング言語で作成したアプリケーションに存在するバグに関する負債は，開発者自身が書いたコード内に生じたバグへの言及が多いのに対して，Docker では，Docker イメージの開発者が実装した部分のバグではなく，外部システムに存在するバグへの言及が多いことを明らかにしている．

**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．また，Docker における開発者が技術的負債を解決する際に役立つような知見を得るため，特に負債が解決されている SATD について調査を行う．

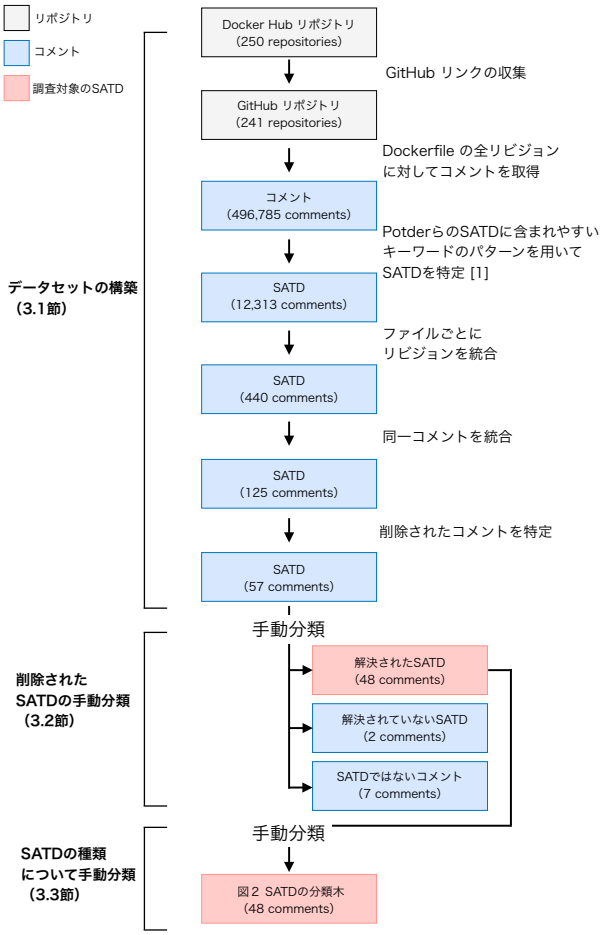


図 1　調査手法の概要

はじめに，Docker における SATD がどの程度解決されているかを調査する（RQ1）．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．よって，解決された SATD の存在期間を調査する（RQ2）．さらに，Azuma[3] の SATD の分類定義に基づいた調査により，解決されている SATD の性質について知見を得ることができる．そのため，どのような種類の SATD が解決されているかを調査する（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．

### 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 の全リビジョンにおけるコメントを調査対象とする．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 9 月）までの全リビジョンの Dockerfile 内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，重複したコメントや SATD ではないコメントが多く含まれる．大量に重複したコメントの統合は計算負荷が高いため，はじめに SATD ではないコメントの除去を行った．本研究では文献 [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．SATD 検出の結果，12,313 件の SATD を含む可能性のある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，コメントごとにリビジョンの統合を行う．また RQ3 にて，SATD が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コメントを追加・削除したコミットの日時の情報を取得する．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．本研究で対象にしたリポジトリおいても，OS やソフトウェアのバージョンごとにディレクトリを作成し，それぞれに Dockerfile を配置しているケースが複数存在した．これらの Dockerfile には同じ 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生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．コメントを統合する Dockerfile を選定する基準として次の 2 点を用意した：(1) SATD のコメントが同じであり，かつ，追加日と削除日が同じ，(2) ディレクトリの深さが同じ．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，コメントのみ削除されている場合の 2 通りが存在している．また，負債の削除について調査するため，コメント内の負債を示す箇所のみが削除されている場合も，コメントの削除として判断する．本研究では開発者が Dockerfile から SATD を削除する手法から，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のみ削除されている場合に絞って調査を行う．削除されたコメント抽出の結果，57 件のコメントが得られた．

### 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 の削除時に負債が解決されているか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を解決していないが SATD を削除している場合を除外する．本研究の手動分類は第 1，第 4，第 5，第 6 著者の 4 名の著者によって行った．また，著者間で分類カテゴリと判断基準を調整するため，削除手法の分類と議論を 3 回に分けて行った．手動分類の結果，負債が解決されている SATD は 48 件であった．

#### 3.2.1 Phase 1: 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負債の原因や 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

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%) |

スニペット 1　負債解決の例

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

```
- # install seccomp
- # TODO: switch to libseccomp-dev since dockerinit is gone
+ # 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

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

論し，分類カテゴリと判断基準を作成した．

#### 3.2.2　Phase 2: 25 件の分類

Phase 1 で作成した分類カテゴリをもとに，ランダムに抽出した 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，Phase 1 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　Phase 3: 22 件の分類

Phase 3 では，残りの 22 件の SATD について Phase 2 と同様の方法で分類と議論を行なった．

表 1 に最終的な分類カテゴリとその分類結果を示し，それぞれの分類の代表例をスニペット 1[*3]，スニペット 2[*4]，スニペット 3[*5] 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．

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

### 3.3　SATD の分類定義に基づいた手動分類

解決されやすい SATD がどのような性質を持つか調査するために，3.2 節で取得した 48 件の SATD について，Azuma[3] の分類定義に基づき手動分類を行う．手動分類は第 1，第 4，第 5，第 6 著者の 4 名の著者によって行い，分類とその議論を順に行った．図 2 に分類木と分類結果を示す．図中の SATD というルートから分岐した 7 個のノードがそれぞれ大分類を表し，大分類からさらに分岐したリーフはその大分類を詳細に分類した小分類を表す．また，各ノードの右上にある数字は各分類の SATD の数を表す．さらに，表 2 に本研究で適用した Azuma[3] による分類の定義を示す．

## 4.　調査結果

本章では，3 章で取得した 48 件の負債が解決された 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 が解決されているか？

**目的：** 一つ目の RQ は SATD の解決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 が解決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，Azuma[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 SATD を取得し，リビジョンごとの比較により解決されている SATD を特定した．

**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．3.2 節より，削除されている SATD のうち負債が解決されているものが 48 件であり，SATD ではないコメントが 7 件存在していた．そのため，本研究によって検出した SATD を含む可能性のあるコメントは 118 件であり，負債が解決されている割合は 40.7%であった．

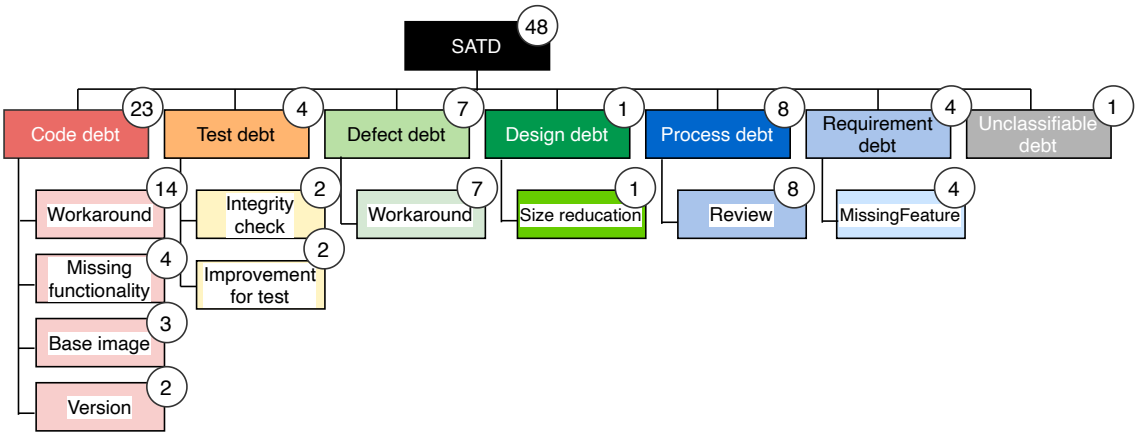


図 2　本研究で用いた SATD の分類木

表 2　SATD の種類の分類定義

| 略称 | 大分類 | 小分類 | 定義 |
|---|---|---|---|
| Code/Workaround | Code debt | Workaround | 次善策での実装に関する負債 |
| Code/Missing functionality | Code debt | Missing functionality | コンテナ内部の処理の欠如に関する負債 |
| Code/Base image | Code debt | Base image | 利用ベースイメージに関する負債 |
| Code/Version | Code debt | Version | 特定のバージョンへの固定に関する負債 |
| Test/Integrity check | Test debt | Integrity check | 真正確認に関する負債 |
| Test/Improvement for test | Test debt | Improvement for test | テストの改善に関する負債 |
| Defect/Workaround | Defect debt | Workaround | 外部システムの既存バグ回避に関する負債 |
| Design/Size reduction | Design debt | Size reduction | Docker イメージのサイズ削減に関する負債 |
| Process/Review | Process debt | Review | Dockerfile のレビューに関する負債 |
| Requirement/Missing Feature | Requirement debt | Missing Feature | 外的に観測できる処理や機能に関する負債 |
| Unclassifiable | Unclassifiable debt | | 明らかに負債ではあるが分類不可な負債 |

特定した 118 件の SATD のうち，48 件（40.7%）の負債が解決されていた．

### 4.2　RQ2: SATD の存在期間はどの程度か？

**目的：** 二つ目の 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：** 解決されている SATD を対象にし，SATD の追加時と解決時の日付情報を取得した．日付は分単位まで取得でき，追加時と解決時の日付の差を 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：** 解決されている 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解決された 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 日未満で解決されていることがわかる．

また，スニペット 4[*6] に本研究で特定した SATD のうち解決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 が解決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

[*6] https://github.com/docker-library/drupal/commit/71ffa1cde017936514d24d64127fb78956ea5c1b

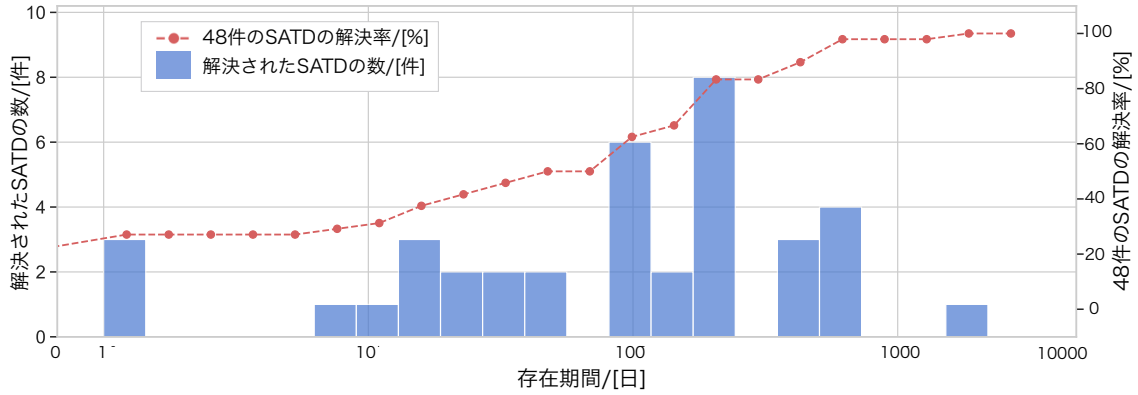


図 3　解決された SATD の存在期間 [日]

スニペット 4　解決までの期間が長い 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　解決までの期間が短い SATD の例

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

なることが考えられる．

一方，スニペット 5[*7] に解決までの期間が短い 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，追加してから解決するまでの期間が短くなる可能性が高い．

SATD が追加されてから解決されるまで，中央値で 67 日，平均値で 166 日の期間があった．

### 4.3　RQ3: どのような種類の SATD が解決されているか？

**目的：** 三つ目の RQ は SATD の種類についてである．解決されている SATD の種類を知ることは，Docker における開発の実態を把握する一つの指標になると考えられる．また，SATD 解決の方法や傾向は，その種類によって大きく異なることが考えられるため，SATD の種類を調査し，開発者による SATD の解決を支援するために有益な知見を獲得する．

**アプローチ：** 解決されている 48 件の SATD について，Azuma[3] の分類定義に基づき，著者 4 名にて手動分類を行った．

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

**結果：** 図 2 より，次善策での実装に関する負債である Code/Workaround が最も多く，次にレビューを求める負債である Process/Review が多く，そして外部システムに存在するバグ回避に関する負債である Defect/-Workaround が多かった．Process/Review が多く存在する理由として，Docker は 2013 年に公開された比較的歴史の浅い技術であるため，開発の知見が少ないことや，特定の開発者に頼った開発が行われていることが考えられる．また，Code/Workaround や Defect/Workaround には，外部システムに起因する次善策の実装に関する SATD が多く存在した．実際に，そのような SATD の多くは外部の修正時に解決されているものが多く，Dockerfile 内に負債の原因がある SATD に比べて，解決の条件が明確であると考えられる．それ以外の Code/Workaround に分類されている SATD には，より良い手法があればその手法に変更したいという内容が記載されているものが複数存在していた．そのため，これらの SATD の解決方法を詳細に調査することで，ベストプラクティスなど公式な文書に最適な実装手法について示すことができる可能性があり，Docker 開発者の支援に繋がると考えられる．

Dockerfile 自体のレビューを求める SATD や，外部システムに起因する SATD が解決されていた．

## 5.　考察

本稿では，Docker における SATD の削除について調査を行った．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，はじめに従来研究との比較を行い，調査結果について考察する．また，本研究結果の開発者支援への応用について述べる．さらに，本研究結果が開発者，研究者のそれぞれに対してどのように

表 3　SATD の負債解決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 |
|---|---|---|
| 解決されている割合 | 40.7% | 74.4% |
| 存在期間の中央値 | 67.0 日 | 18.2〜172.8 日 |

役立つかを説明する．

### 5.1　従来研究との比較

本研究での RQ1 および RQ2 の調査結果，および Java プロジェクトを対象に行われた Maldonado ら [4] による研究結果を表 3 に示す．表 3 より，Docker における SATD は，Java と比較して解決された SATD の割合が 33.7%低いことが明らかになった．それにより，Docker における SATD は解決が困難なものが多い可能性がある．

また，RQ3 の調査結果について，Azuma[3] による SATD の種類に関する調査では，真正確認に関する負債である Test/Integrity check が全体の 12.9%を占めていたのに対して，本研究においては 4.7%のみであった．Azuma[3] により，真正確認に関する負債は Docker 固有のものであると主張している．本研究でも当該負債の解決数が少ないことを示していることから，開発の知見が少ない Docker において，Docker 固有の SATD の解決が容易でないことを示唆している．今後，Docker 固有の SATD について解決方法や負債の原因を詳細に調査することで，Docker 固有の問題解決への糸口になることが期待できる．

### 5.2　開発者支援への応用

本研究結果より，解決されている SATD には外部システムに起因するものが多いことを明らかにした．そのような SATD を解決するために，開発者は原因となる外部システムを注視し，修正後速やかに対応する必要がある．しかし，それらの SATD の中には，外部が変更されてから 360 日後に解決されているものも存在していた．そのため，外部変更時に開発者に対して通知を行うことができれば，より早期に SATD を解決できると考えられる．

本研究では，外部システムに起因する SATD の早期開発を開発者に対して促すため，外部システムの変更を検知し，開発者に通知を行うツールを作成した．ツールについての概要を次章で述べる．

### 5.3　開発者と研究者への貢献

#### 5.3.1　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 にも，エラーメッセージのみ記載されており修正箇所や詳細がわかりにくいケースが存在した．そのため，Docker における SATD の解決率が低い原因の一つとして，負債の詳細が書かれていないことが考えられる．開発者は，コメント内に詳細や原因箇所のリンクを挿入するなどの対策をすることで，SATD を解決するまでの期間短縮が可能になると考えられる．

スニペット 6　外部システムに起因する SATD の例

```
1 # TODO: the distribution given by apache is not gzipped
        correctly - unzip correctly when fixed
```

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．

#### 5.3.2　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 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.　開発者支援ツール

### 6.1　ツールの動作対象となる SATD の手動分類

はじめに，開発者支援ツールの動作対象となる外部システムに起因する SATD を取得するため，48 件の SATD を対象に第 1 著者によって手動分類を行った．分類の結果，外部システムに起因する SATD は 12 件であった．代表的な例をスニペット 6 に示す．スニペット 6 では，Apache によって配布されているファイルが正しく圧縮されていないことを記載しており，負債の原因が明らかに外部に存在すると考えられる．

また，取得した 12 件の SATD のうち，8 件が GitHub のリポジトリのリリースバージョンや Issue について言及していた．これらの SATD の 7 件は，実際に対象のリポジトリで新しいバージョンがリリースされた後，もしくは Issue がクローズされた後に解決されていた．

スニペット 7　ツールの動作対象となるコメントの例

```
1 # 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)
```


図 4　ツールにより作成される Pull Request の例

### 6.2　ツール詳細

本研究では，開発者支援のための初期段階のツールとして，GitHub のリリースバージョンや Issue の変更を検知し，開発者に通知を行うツールを作成した．開発者支援ツールの作成には，GitHub が提供する GitHub Actions を利用した．ツールは GitHub 上で公開しており，開発者は GitHub 内の自身のリポジトリに設定用のファイルを 1 つ作成するだけで利用できる．ツールは開発者が設定した時刻に動作し，Dockerfile 内にコメントされている GitHub リポジトリのリリース情報を示すリンク，特定の Issue を示すリンクを自動で検知する．実行時の 24 時間以内に，検知したリンク先のリポジトリで新しいバージョンがリリースされる，もしくはリンク先の Issue がクローズされている場合，Dockerfile 内の該当のコメント上部に解決を促すコメントを挿入し，Pull Request を作成する．スニペット 7 に，ツールの動作対象となる特定の Issue を示すリンクを含んだコメントの例を示し，図 4 に，リンク先のリポジトリで新しいバージョンがリリースされた際に作成される Pull Request の例を示す．ツールの導入により開発者は，最大 24 時間以内に外部システムの変更を知ることができ，SATD の早期解決を促すことができる．

## 7.　妥当性への脅威

### 7.1　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 Dockerfile を取得できていない可能性がある．

また，本研究で使用した SATD 検出手法の適合率が 86.4%であることから，3 章で取得したコメントには，SATD ではないコメントが含まれる可能性がある．そのため特定したコメントに対して，今後目視分類を行うべきである．

### 7.2　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．

## 8.　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 の負債解決割合やその種類，また解決されるまでの期間について分析を行った．その結果，Dockerfile に含まれる SATD の 40.7%が解決されていることを明らかにした．SATD が解決されるまでの期間は，中央値が 67 日で平均値が 166 日であった．また，Dockerfile 自体のレビューを求める SATD や，外部システムに起因する SATD が多いことを明らかにした．調査結果を従来研究の結果と比較して，Docker における SATD は解決が困難なものが多い可能性を示し，Docker における SATD の性質調査の重要性を明らかにした．さらに，開発者に対して SATD の早期解決を促すための支援ツールを作成し，GitHub 上で公開した．

本研究において，3.2 節で行った SATD 解決の分類は，負債が解決済みか否かという点に着目しており，その解決手法の分類は行っていない．より詳細な解決手法の分類は，Docker における SATD の解決に関するさらなる知見となり，開発者支援ツールの発展にもつながるため，今後調査する必要があると考えられる．また，Dockerfile はコンテナイメージの構成を記述するものであり，技術的負債がソフトウェアの品質や開発に与える影響が一般的なプログラミング言語と異なる可能性がある．そのため，Docker における SATD が与える影響について詳細

に調査する予定である．さらに，Docker における平均的な開発者数や更新頻度と本研究結果の関連性を調査し，Docker における SATD 解決の現状についてより詳細に調査する．

## 謝辞

本研究の一部は JSPS 科研費 JP21H04877, 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 *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 *Proceedings of International Conference on Software Analysis, Evolution, and Reengineering*, pp. 179–188, 2016.

[3] H. Azuma. An empirical study on self-admitted technical debt in container virtualization. Master's thesis, Graduate School of Information Science and Technology Osaka University, 2021.

[4] E. D. S. Maldonado, R. Abdalkareem, E. Shihab, and A. Serebrenik. An empirical study on the removal of self-admitted technical debt. 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 *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 *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 *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 *Proceedings of International Conference on Software Engineering*, p. 251–260, 2008.