特別研究報告

題目

Docker イメージの品質改善を目的とした Dockerfile 最適化支援ツール

指導教員

楠本 真二 教授

報告者

森内 涼太

令和 7 年 2 月 6 日

大阪大学基礎工学部情報科学科

令和 6 年度 特別研究報告

Docker イメージの品質改善を目的とした Dockerfile 最適化支援ツール

森内 涼太

内容梗概

軽量なコンテナ型仮想環境プラットフォームとして Docker が注目されている．Docker ではコンテナを作成するために Docker イメージを用いる．Docker イメージは Docker Hub などのホスティングサービスを通じて共有が可能であり，仮想環境の再現性や可搬性の高さに貢献している．Docker イメージの品質改善としては，Dockerfile から生成される Docker イメージのサイズとビルド時間の削減が重要である．しかしながら，Dockerfile の最適化作業に必要なビルド時の動的情報は，コードエディタ上の静的情報であるソースコードからは把握できない．そのため，動的情報は Dockerfile のエディタとは独立したターミナル上で確認する必要があり，Dockerfile の各行との対応が取りづらい．また，最適化効果の把握に必要なビルド間の差分を直接把握する方法は存在しない．本研究では Docker イメージの品質改善を目的とした Dockefile 最適化支援ツールを提案する．提案ツールはビルド時の動的情報を解析し，解析結果をコードエディタ上で閲覧できる機能を提供する．本研究では提案ツールの有効性を評価するために被験者実験を実施した．被験者実験の結果，Docker イメージのサイズ削減を目的とした最適化作業に対して，正確性と効率性の向上に一定の効果があることを確認した．一方で，提案ツールには改善の余地があることも判明した．被験者実験の結果を踏まえ，一部の機能に対して改善を行った．

主な用語

Docker，Docker イメージ，Dockerfile，最適化，レイヤ，ビルドキャッシュ

目次

図目次

表目次

1　はじめに

軽量なコンテナ型仮想環境プラットフォームとして Docker が注目されている．Docker は，一般的なハイパーバイザ型の仮想化技術と比べてリソース効率が高く，仮想環境の迅速なデプロイとスケーリングを可能とする [1]．コンテナの作成に用いる Docker イメージは Docker Hub などのホスティングサービスを通じて共有が可能であり，仮想環境の再現性や可搬性の高さに貢献している [2]．このような利点から，Docker は OSS のようなソフトウェア開発プロジェクトで広く採用されている [3]．さらにソフトウェア開発のみならず，IoT やクラウドコンピューティングなど幅広い分野で活用されている [4][5]．

Docker イメージの開発においては，一般的なソフトウェアの開発と同様，品質改善が欠かせない．具体的には，Docker イメージに対するサイズとビルド時間の削減が挙げられる [6]．前者はリソース効率や可搬性の向上が，後者は Docker イメージの開発効率の向上が目的である．Docker イメージの品質改善には，Docker イメージのソースコードである Dockerfile の修正が必須である．本稿では，このような品質改善を目的とした Dockerfile の修正を最適化と表現する．

Dockerfile の最適化は，一般的なソースコードに対するバグ修正やリファクタリングと同様，容易ではない．本研究では，Dockerfile の最適化作業を「問題点の発見」「問題点の分析」「問題点の修正」「修正効果の把握」の 4 ステップに分解して考える．「問題点の発見」では，サイズが大きい，もしくはビルドキャッシュが利用されていないなどの問題を抱えたレイヤを発見するために，ビルド時の動的情報を把握しなければならない．一方で，Dockerfile の開発に用いるコードエディタで把握できる情報はソースコード上の静的情報に限られている．コードエディタとは独立したターミナル上で動的情報を調べる場合，動的情報と Dockerfile の各行の対応が取りづらい．また「修正効果の把握」では，動的情報が Dockerfile の修正前後でどのように変化したか把握しなければならない．

本研究の目的は，Dockerfile 最適化作業の支援である．特に最適化作業の 4 ステップのうち，「問題点の発見」と「修正効果の把握」に着目する．そのためにビルド時の動的情報を解析し，解析結果をコードエディタ上で閲覧できるツールを提案する．本研究では，提案ツールの有効性を評価するために被験者実験を実施した．実験の結果，Dockerfile の最適化作業において，正確性と効率性の向上に一定の効果があることを確認した．

2　準備

2.1　Docker イメージの開発

Docker イメージの構築手順は，Dockerfile と呼ばれるソースコード内に記述される．Dockerfile の記述例を 図 1 に示す．図の Dockerfile には，Python プログラムの実行環境を構築する手順が記述されている．Ubuntu の Docker イメージをベースにパッケージや依存ライブラリのインストールを行い，Python プログラムをコンテナ起動時に実行できる環境を用意している．

Docker のビルドコマンドを実行すると，Dockerfile に記述されたコマンドが上から順に実行され，各コマンドごとにレイヤが生成される．これら複数のレイヤによって 1 つの Docker イメージが構成される．なお，各レイヤは一度生成されると，変更されるまでビルドキャッシュされる．ビルド時に変更のないレイヤは再構築せずにビルドキャッシュが利用されるため，ビルド時間の短縮に繋がる．

2.2　Dockerfile の最適化

Docker イメージの品質を左右するイメージのサイズとビルド時間を削減するためには，Dockerfile の最適化が必須である．Dockerfile の最適化は Docker プロジェクトで一般的に実施されており，その注目度は高い [7]．Docker Hub に公開されている Docker イメージのサイズは，最適化の実施により，バージョンを重ねるにつれて減少傾向にあると報告されている [8]．GitHub で管理されている Docker プロジェクトのリポジトリでは，Dockerfile に関連するコミットに，単なるリファクタリングのみならずイメージサイズやビルド時間の削減を目的とした最適化が多く含まれていると確認されている [9]．

本研究では，Dockerfile の最適化作業を「STEP1：問題点の発見」「STEP2：問題点の分析」「STEP3：

```
1  # ベースイメージを指定
2  FROM ubuntu:20.04
3  # パッケージをインストール
4  RUN apt-get update && apt-get install -y \
5      python3 python3-pip \
6      openjdk-17-jdk \
7      && rm -rf /var/lib/apt/lists/*
8  # 依存ライブラリをインストール
9  COPY . .
10 RUN pip3 install --no-cache-dir -r requirements.txt
11 # コンテナ起動時にPythonプログラムを実行
12 CMD ["python", "sample.py"]
```

図 1　Dockerfile の記述例

問題点の修正」「STEP4：修正効果の把握」の 4 ステップに分解して考える．ここでは前述の 図 1 を例に説明する．まず各レイヤのサイズを調べた結果，4〜7 行目の RUN コマンドに対応するレイヤのサイズが，他のレイヤと比較して大きいと判明したとする．次に問題となっている RUN コマンドを分析すると，Python プログラムの実行環境の構築とは無関係な openjdk-17-jdk がインストールされていると分かる．つまり，不要なパッケージがレイヤのサイズに影響を与えている可能性が高い．この分析結果をもとに 6 行目を削除し，改めて Docker イメージのビルドを行う．最後に，RUN コマンドのレイヤのサイズを修正前後で比較し，修正効果がどの程度得られたか把握する．

2.3 Dockerfile 最適化の課題

2.2 節で述べた Dockerfile の最適化作業について，「STEP1：問題点の発見」と「STEP4：修正効果の把握」には課題が存在する．

「STEP1：問題点の発見」を実施する際には，各レイヤのサイズやビルド時間の変動，ビルドキャッシュがどのレイヤまで利用されているかといった動的情報を把握しなければならない．一方で，Dockerfile の開発に用いるコードエディタで把握できる情報はソースコード上の静的情報に限られるため，問題点の発見は困難である．また，動的情報を調べる Docker の標準コマンドや dive[*1]といったツールは存在するが，コードエディタとは独立したターミナル上で調べなければならない．そのため，動的情報と Dockerfile の各行の対応が取りづらく，Dockerfile の修正箇所の特定を妨げる要因となる．

「STEP4：修正効果の把握」を実施する際には，STEP1 で調べた動的情報が Dockerfile の修正前後でどのように変化したか把握しなければならない．しかしながら，このようなビルド間の差分を直接把握する方法は筆者の知る限り存在しない．そのため，開発者が自ら前後のビルド結果を比較して差分を計算する必要がある．またビルド間の差分は，STEP1 でビルドキャッシュ効果の有無によるビルド時間の変動を把握する場合にも用いるため，最適化作業に欠かせない情報であるといえる．

[*1] https://github.com/wagoodman/dive.git

3　提案手法

3.1　概要

2.3 節で述べた課題解決のため，提案ツールの機能要件として以下の 4 点を定めた．

R1.　サイズが大きいレイヤを特定できる

R2.　ビルドキャッシュが利用されていないレイヤと，そのレイヤによって生じたビルド時間の変動を把握できる

R3.　Dockerfile の修正によって得られた効果を把握できる

R4.　問題のレイヤに対応する Dockerfile のソースコード行（修正箇所）を特定できる

本研究ではこれらの要件を満たす提案ツールを，コードエディタの拡張機能として試作した．コードエディタには，開発者の間で高いシェアを誇る[*2]Visual Studio Code（VS Code）を採用した．以降では提案ツールの機能について順に説明する．

3.2　ビルドと解析結果の描画

提案ツールは VS Code の拡張機能として提供される．図 2 に示すように，VS Code 上に Dockerfile を開いた状態で画面上部の DfileProfiler ボタンをクリックすると，自動で Docker イメージをビルドし，イメージの解析結果を右側のペインに表示する．イメージの解析結果は相対表示画面と差分表示画面で構成されており，画面の切り替えはタブで行う．各画面の詳細は続く 3.3 節と 3.4 節で説明する．

3.3　相対表示画面

図 2 の右側のペインに示すように，相対表示画面では，各レイヤのサイズとビルド時間を棒グラフで表現している．グラフの長さからサイズの大きいレイヤが直感的に分かるため，要件 R1 を満たす狙いがある．さらにサイズの大きいレイヤから順に調べ，Dockerfile の最適化効果が大きい順に最適化を実施できる狙いもある．パレートの法則に当てはめると，Docker イメージのサイズを過剰に大きくさせている 8 割の原因がサイズの大きい上位 2 割のレイヤに集中しているといえるためである．パレートの法則に基づく最適化手法はソフトウェア工学において一般的である．Russell ら [10] はソフトウェアの最適化において，プロファイリングツールの解析から最大のボトルネックを特定し，上位の問題解決のみに集中するというパレートの法則に基づいた手法を提案している．

また，相対表示画面のビルド時間の欄について，ビルドキャッシュが利用されているレイヤには，ビ

[*2] https://survey.stackoverflow.co/2024/technology#1-integrated-development-environment

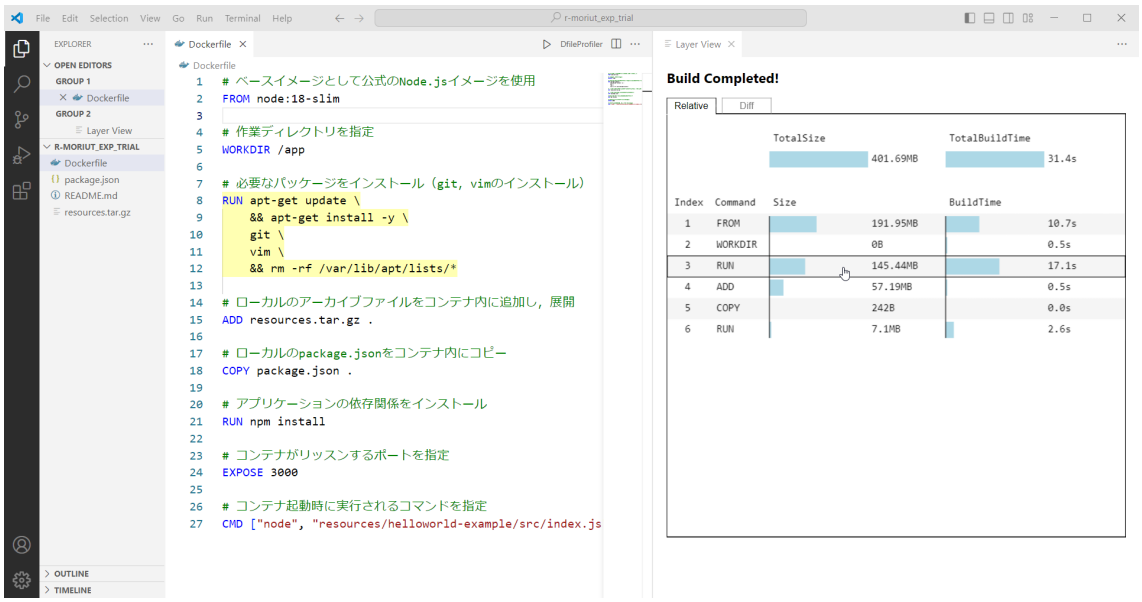

図 2　提案ツールの全体像

ルド時間の値のかわりに “CACHED” という文字が表示される設計にした．この設計によって，どのレイヤからビルドキャッシュを利用していないかが一目で判断できるため，要件 R2 の「ビルドキャッシュが利用されていないレイヤを把握できる」という要素を満たす狙いがある．

3.4　差分表示画面

図 3 に示すように差分表示画面では，各レイヤのサイズとビルド時間について，1 つ前のビルド結果との差分を棒グラフで表現している．直前のビルド結果と比べて増加している場合は赤色のグラフ，減少している場合は緑色のグラフとなる．図 3 の場合，6 番目の RUN コマンドのレイヤはサイズが 10.0MB 増加し，1 番目の FROM コマンドのレイヤはビルド時間が 10.7s 減少したと分かる．グラフの色や長さ，向きなどでどのレイヤのサイズとビルド時間がどれほど増減したかが直感的に分かるため，要件 R2 の「ビルド時間の変動を把握できる」という要素と，要件 R3 を満たす狙いがある．

3.5　レイヤと Dockerfile のマッピング

前述の 図 2 に示すように，解析結果のレイヤにマウスオーバーすると，Dockerfile の対応する行がハイライトされる．また，レイヤをクリックすると Dockerfile の対応する行にジャンプする．このようなレイヤと Dockerfile のマッピング機能によって，レイヤと Dockerfile の各行の対応関係が把握しやすくなるため，要件 R4 を満たす狙いがある．

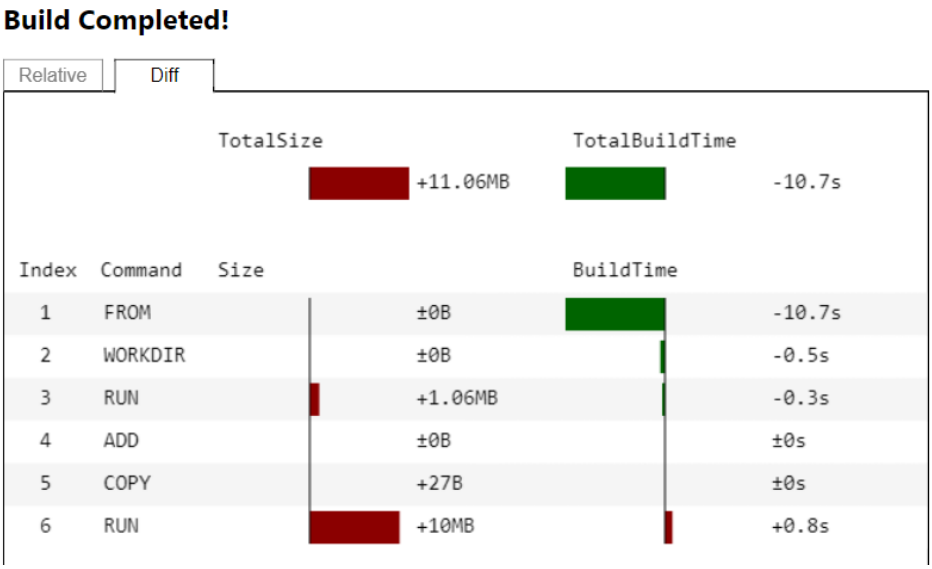

図 3　差分表示画面

3.6　再構築対象のレイヤの通知

Dockerfile や外部の依存ファイルを編集すると，ビルドキャッシュが利用できなくなるレイヤに再構築を示すアイコンが表示される．アイコンの表示と非表示は編集の度にリアルタイムで反映される．たとえば 図 4 に示すように，package.json の中身を編集すると，解析結果における COPY コマンド以降のレイヤにアイコンが表示される．このような再構築対象のレイヤの通知機能によって，どのレイヤからビルドキャッシュを利用できなくなるかがビルド前に確認しやすくなるため，要件 R2 の「ビルドキャッシュが利用されていないレイヤを把握できる」という要素を満たす狙いがある．

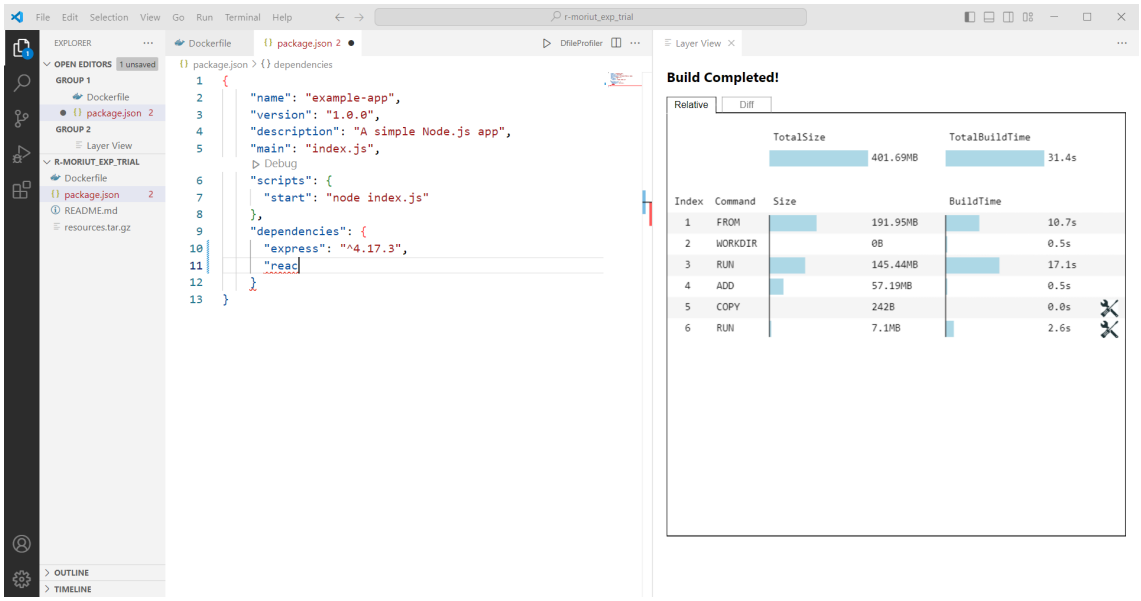

図 4　再構築対象のレイヤが通知された様子

4　実験

4.1　概要

提案ツールの有効性を評価するために被験者実験を実施した．実験では，Dockerfile の最適化タスクを被験者に取り組ませる．被験者は提案ツール使用の有無で 2 グループに分けた．評価指標には正解率と事後アンケートの回答結果を用いた．

4.2　被験者とグループ分け

本実験の被験者は大阪大学大学院情報科学研究科に所属する修士の学生 9 名，大阪大学基礎工学部情報科学科に所属する学部生 3 名の計 12 名である．

まず，被験者全員に対して事前アンケートを行った．アンケートでは，ドレイファスモデル [11] を元に Docker の習熟度を「初心者」「中級者」「上級者」「熟練者」「達人」の 5 段階に区別し，被験者自身が最も当てはまると思う習熟度を選択する形式にした．

次に，事前アンケートの回答結果に基づいて，被験者を「提案ツール使用」と「提案ツール不使用」の 2 グループに分けた．グループ別の Docker の習熟度の分布を 図 5 に示す．グループ間での能力の偏りを防ぐため，Docker の習熟度の分布がなるべく同じになるよう振り分けた．

なお，「提案ツール使用」グループの被験者に対しては，本実験の前にツールの使い方を 30 分ほど学習させた．被験者間で提案ツールの使用に慣れるまでの差を排除し，提案ツールの有効性をより正確に評価できるようにするためである．学習用に用意した簡潔な Dockerfile を対象に，3.2 節〜3.6 節で述べた各機能を提案ツールの操作を通して確認させた．

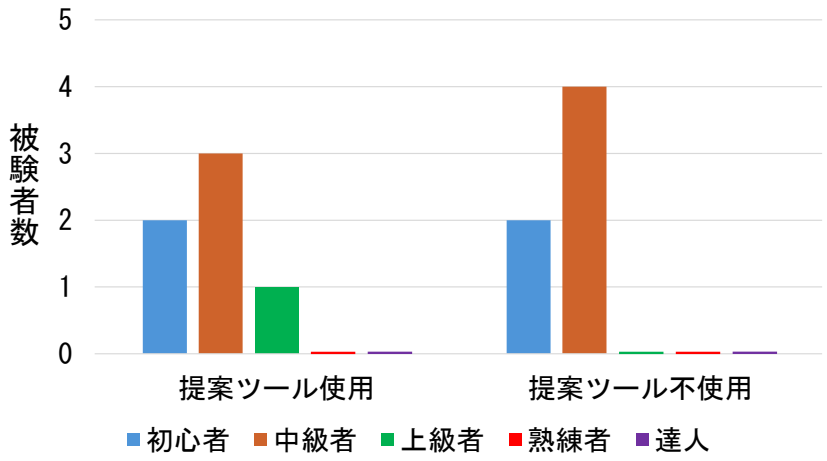

図 5　グループ別の Docker の習熟度の分布

4.3 Docker の事前学習

被験者全員に対して，本実験の前に 60 分ほど Docker の事前学習を行った．被験者の Docker への習熟度が提案ツール使用者の想定レベルを満たすようにすることで，より公正な評価ができるようにするためである．前述の 図 5 に示すように，被験者の 30% は初心者で Docker を全く知らない状態である．対して，提案ツールの想定レベルは中級者以上である．具体的な想定レベルとして，Docker の基本的な概念を理解しており，Docker の公式ドキュメント[*3]を参考に Dockerfile の作成や Docker イメージのビルドができる状態が望ましい．このような想定レベルを踏まえ，Docker に関する基本的な用語と，実験タスクに取り組むために必要な知識を学習内容に取り入れた．

4.4 タスク

4.4.1 概要

実験タスクは Dockerfile の最適化である．タスクの設計においては，Dockerfile を用いた Web フロントエンドの開発を想定した．Web 開発は Docker のユースケースでよく見られる．Haque ら [12] は Stack Overflow 上の Docker に関連する投稿を調査し，Web サーバと Web フレームワークの設定や WebDriver の使い方など，Web 開発に関連する質問が多数見られたと報告している．このように Web 開発をシナリオに設定することで，提案ツールが実用的な場面においても有効であることを確認する．なお，タスクは 2 種類に分けられる．Docker イメージのサイズ削減を目的とした最適化タスクと，ビルド時間の削減を目的とした最適化タスクである．

4.4.2 イメージサイズ削減タスク

本タスクは「Q1：問題点の発見」「Q2：問題点の分析」「Q3：問題点の修正」「Q4：修正効果の把握」の 4 つの設問で構成される．各設問の内容を 表 1 に示す．題材の Dockerfile には，パッケージリストの削除忘れや不要なファイルのコピーなどイメージサイズを大きくさせる問題点が複数含まれている．被験者には問題点それぞれについて，30 分間で Q1〜Q4 を繰り返し実施させた．

4.4.3 ビルド時間削減タスク

本タスクは 4.4.2 節同様，「Q1：問題点の発見」「Q2：問題点の分析」「Q3：問題点の修正」「Q4：修正効果の把握」の 4 つの設問で構成される．各設問の内容を 表 2 に示す．題材の Dockerfile には，ビルドキャッシュの不十分な活用といったビルド時間を増加させる問題点が 1 つ含まれている．被験者にはその問題点について，15 分間で Q1〜Q4 を実施させた．

[*3] https://docs.docker.com/

表 1　イメージサイズ削減タスクにおける各設問の内容

項目	内容
Q1：問題点の発見	サイズが大きいレイヤと，そのサイズの値を答えてください．
Q2：問題点の分析	Q1 のレイヤは，なぜサイズが大きいのでしょうか．その要因を答えてください．
Q3：問題点の修正	Q2 の要因について，改善策を考えて Dockerfile を修正してください． 修正が完了したら，どのように修正したか答えてください．
Q4：修正効果の把握	Q3 の修正によって，効果がどれくらい得られたか答えてください．

表 2　ビルド時間削減タスクにおける各設問の内容

項目	内容
Q1：問題点の発見	index.html の中身を資料記載の内容（ここでは省略）に丸ごと差し替えてください． 差し替え後 Docker イメージをビルドしてビルド時間がどれだけ増加したか答えてください．
Q2：問題点の分析	Q1 のファイルの編集でなぜビルド時間が増加したのでしょうか．その要因を答えてください．
Q3：問題点の修正	Q2 の要因について，改善策を考えて Dockerfile を修正してください． なお，今後 index.html を含めた Web サイトのリソースを更新する頻度は極めて高いと想定してよいです． 修正が完了したら，どのように修正したか答えてください．
Q4：修正効果の把握	index.html について資料記載の内容（ここでは省略）に変更してください． 変更後 Docker イメージをビルドして，Q3 の修正によって効果がどれくらい得られたか答えてください．

4.5　評価指標

実験の評価指標には，正解率と事後アンケートの回答結果を用いる．

正解率とは全体の設問数に対して正解した設問数の割合である．被験者ごとにタスクの各設問の解答を採点して正解率を算出し，「提案ツール使用」グループと「提案ツール不使用」グループのそれぞれで平均正解率を算出した．

また，「提案ツール使用」グループの被験者 6 名に対して事後アンケートを行った．アンケートはリッカート尺度 [13] での 5 段階評価で回答する質問と，自由記述で回答する質問で構成される．5 段階評価で回答する質問では，機能要件で定めた問題点の発見，修正効果の把握，レイヤと Dockerfile のマッピングに提案ツールが役立ったか尋ねたほか，直感的な操作や動作の軽さなど，提案ツールが満たすべき非機能要件についてどのように感じたか尋ねた．自由記述で回答する質問では，提案ツールの良かった点や改善すべき点を尋ねた．

5　結果と考察

5.1　イメージサイズ削減タスクの平均正解率

イメージサイズ削減タスクの平均正解率を 図 6 に示す．図を見ると，いずれの設問も提案ツールを使用した方が平均正解率がおよそ 3〜13% 高くなっている．特に Q4 の平均正解率については，およそ 2 倍の改善が見られた．

「Q1：問題点の発見」と「Q4：修正効果の把握」の平均正解率が向上した理由としては，Q1 では提案ツールの相対表示画面の機能が，Q4 では差分表示画面の機能が有効に働いたためと考えられる．さらに Q4 の解答を分析すると「提案ツール不使用」グループの被験者からは，適切な差分値からかけ離れた明らかに計算ミスの可能性が高い解答が散見された．このような計算ミスを防ぐ提案ツールの強みが平均正解率の向上に貢献したともいえる．

また，「Q2：問題点の分析」と「Q3：問題点の修正」の平均正解率が向上した理由としては，Q1 と Q4 に解答する効率が良くなった結果，副次的に Q2 と Q3 に取り組む時間が増えたためと考えられる．

一方で，提案ツール使用の有無に関わらず平均正解率は最高でおよそ 25〜30% であり，全体を通して低い結果となっている．これは最適化で実施する一連の流れに被験者が慣れておらず，被験者のレベルに対してタスクの難易度が上がってしまった点が原因だと考えられる．

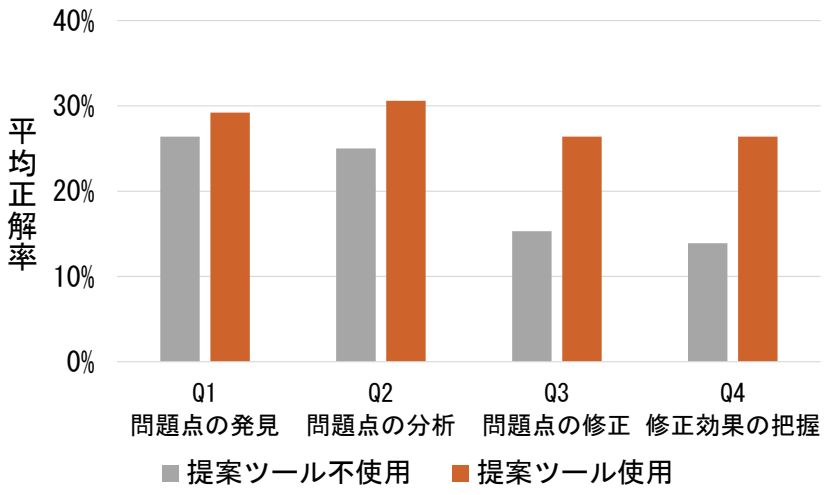

図 6　イメージサイズ削減タスクの平均正解率

5.2　ビルド時間削減タスクの平均正解率

ビルド時間削減タスクの平均正解率を 図 7 に示す．図を見ると，Q1 と Q3，Q4 は提案ツールを使用した方が平均正解率がおよそ 8〜16% 低くなっている．

「Q1：問題点の発見」と「Q4：修正効果の把握」の平均正解率が向上しなかった理由としては，提案ツールの差分表示画面の機能が有効に働かなかったためと考えられる．要因として，ビルド時間の履歴を提案ツール上で遡って確認できなかった点が大きい．本タスクではビルドキャッシュ効果の有無によるビルド時間の変動を観察しなければならない．提案ツールではビルド結果の差分は直近との比較に限られているが，ターミナルではウィンドウを閉じない限りビルド結果を遡って確認できる．外部ファイルの編集ミスや Dockerfile の修正ミスによってビルドの試行錯誤を繰り返してしまうと，提案ツールではビルド時間の変遷を追えなくなるという欠点があった．

また，「Q2：問題点の分析」と「Q3：問題点の修正」の平均正解率が向上しなかった理由としては，Q1 に解答する効率が悪くなった結果，副次的に Q2 と Q3 に取り組む時間が減ったためと考えられる．

5.3　事後アンケート

5 段階評価での回答結果を 図 8 に示す．図の上記 5 つの項目に対して，いずれも被験者の 80% 以上が肯定的な回答を示した．また自由記述での回答では，提案ツールの良かった点について 4 名の被験者が差分表示画面の機能を挙げていた．他にも 2 名の被験者が相対表示画面の機能を，1 名の被験者がレイヤと Dockerfile のマッピング機能と再構築対象のレイヤの通知機能を挙げていた．一方で 3 名の被験者が「直近との差分だけではなく，ビルド結果の差分を遡って閲覧できる機能がほしい」といった意見

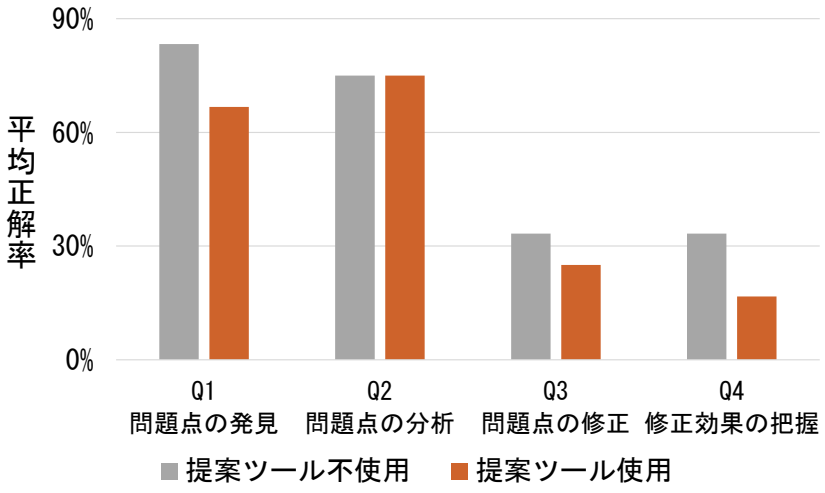

図 7　ビルド時間削減タスクの平均正解率

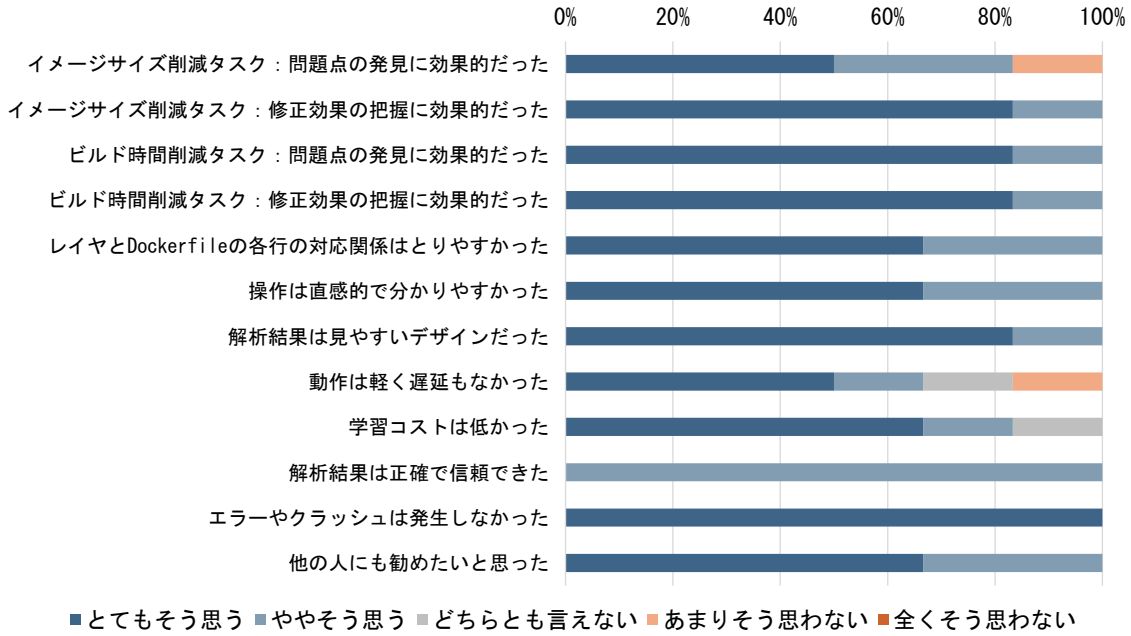

図 8　提案ツールに関する事後アンケートの 5 段階評価での回答結果

を挙げており，5.2 節で述べたビルド時間削減タスクで「提案ツール使用」グループの平均正解率が低下した要因がここからも確認できる．

他方で 図 8 を見ると，「動作は軽く遅延もなかった」という項目に対して被験者のおよそ 30% が否定的な回答をしている．自由記述での回答では，改善すべき点について 1 名の被験者が「描画処理が長すぎる」といった意見を挙げており，被験者の環境によって提案ツールの処理時間に大きな差異が出てしまったと考えられる．

6　提案ツールの改善

5 節で述べた実験結果と考察をもとに，提案ツールの改善を行った．以降では改善した機能について順に説明する．

6.1　差分比較対象の選択

差分表示画面に，比較対象のビルド結果を選択できる機能を追加した．図 9 に示すように，本機能は比較対象のビルド結果を直接選択するボタン（円形ボタン）と，前後のビルド結果に移動できる前後選択ボタン（‘<’, ‘>’ ボタン）で構成される．一番左の円形ボタンが直近のビルド結果を表しており，右に進むにつれてビルド結果を遡ることができる．最新のビルド結果に対して，直近のみならず任意のビルド結果との差分を確認できるため，ビルド時間削減タスクの正解率を向上させる狙いがある．

6.2　コメント入力と表示

6.1 節の機能追加を踏まえ，Docker イメージのビルド時にコメントを残せる機能を追加した．図 10 に示すように，VS Code のサイドバーにあるテキストエリアにコメントを入力して Build ボタンをクリックすると，改善前の提案ツールと同様に自動で Docker イメージをビルドし，イメージの解析結果を右側のペインに表示する．差分表示画面で各円形ボタンにマウスオーバーすると，そのビルドの実行

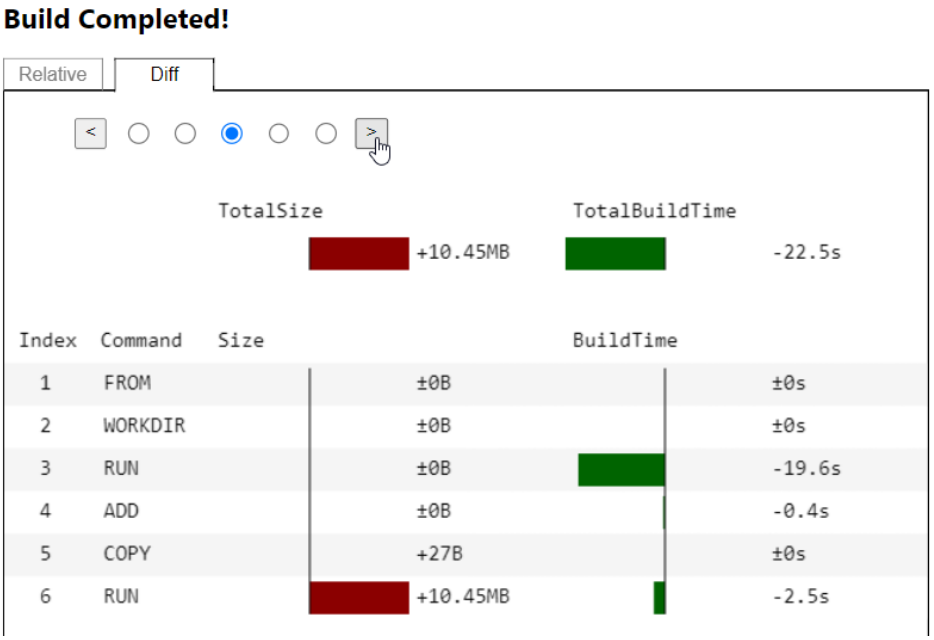

図 9　差分比較対象の選択

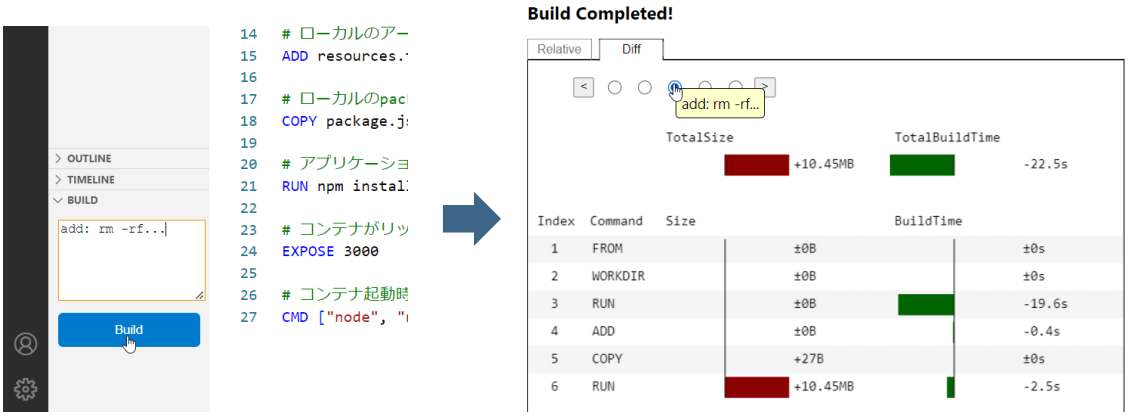

図 10　コメント入力と表示

時にコメントしていた内容が表示される．Git のコミットメッセージのように各ビルド時点で変更した内容や理由を後から確認できるため，差分の比較対象を選択する際に役立つ狙いがある．

7　制約

本研究で試作した提案ツールには制約が存在する．

最適化作業の 4 ステップうち，提案ツールが支援するステップは限定されている．本研究では「STEP2：問題点の分析」と「STEP3：問題点の修正」については着目していない．提案ツールで問題点を発見した後は，Dockerfile のベストプラクティス[*4]などを参考にして自力で問題点の分析と修正を行う必要がある．そのため問題点を発見しても要因の解明や Dockerfile の修正ができない可能性がある．

提案ツールを十分に活用するためには，Docker の習熟度について一定のレベルが要求される．解析結果を見る際に，レイヤやビルドキャッシュの概念を理解しておかなければならないためである．4.3 節で述べた通り，被験者実験においても Docker の事前学習を行い，初心者の能力を底上げした．このように，提案ツールを使用する前に自身の Docker の習熟度を把握し，想定レベルに達していない場合は知識を身につけておく必要がある．また本研究の実験では，被験者のレベルが中級者以下に偏っていたため，上級者以上に対するツールの有効性が十分に評価できていない．そのため上級者以上に相当する Docker の開発者が提案ツールを使用した場合，新たな制約が判明する可能性がある．

提案ツールの実用的な場面における有効性に対しても，十分な評価ができているとはいえない．実験タスクのシナリオは Web フロントエンドの開発 1 つに限定していた．Docker のユースケースには，他にアプリケーション開発や CI/CD パイプラインの構築などが挙げられる [12]．このような様々なシナリオ下で提案ツールを使用した場合，新たな制約が生じる可能性がある．

[*4] https://docs.docker.com/build/building/best-practices/

8　関連研究

ソフトウェアの可視化に関する研究は数多く存在する [14]．Lacerd ら [15] や兼光ら [16] は，プログラムやソースコードの解析情報をグラフに可視化することで，リファクタリングの支援に取り組んでいる．Schaad ら [17] は，データのフローやメモリの割り当てをグラフに可視化することで，パフォーマンスの向上を目的とした最適化の支援に取り組んでいる．これらの可視化手法は 2 次元空間上に適用する手法である一方，3 次元空間上に可視化する手法も提案されている．Romano ら [18] は，ソフトウェアのモジュールを都市の建造物で表現し，さらに VR 技術を用いた仮想空間上に都市を再現する手法を提案している．被験者実験の結果，提案手法はプログラム理解の正確性や効率性の向上に効果があったと報告されている．本研究では，Docker イメージの動的情報を解析し，2 次元空間上で棒グラフに可視化するツールを提案した．Romano らのような 3 次元空間を用いた可視化手法を提案ツールに適用する余地がある．

また，Dockerfile の最適化に関する研究も盛んである．Rosa ら [7] や Henkel ら [19]，Durieux ら [20] は，Docker イメージの品質低下に繋がる記述（Dockerfile スメル）に対して，検出と修正を行うために抽象構文木を用いた手法を提案している．この提案手法はソースコード上の静的情報を解析する手法であるため，改善対象とする Dockerfile スメルは，静的解析のみで検出と修正が可能なスメルに限定されている．Xu ら [21] は，静的解析に加えて動的解析を用いた手法を提案している．しかしながら，この提案手法で改善できる Dockerfile スメルは 1 つに限定されている．以上の研究はいずれも，Dockerfile スメルの検出と修正の自動化が目的である．一方で，Dockerfile のベストプラクティスには自動修正が難しく，開発者の判断が求められるものも多い．たとえばインストールするパッケージやコピーするファイルには，Docker イメージのビルド時点では不要だが，コンテナ上での操作や将来的な保守の観点から必要とする場合も考えられる．本研究の提案ツールは，動的情報を参考に開発者が手動で最適化を実施する場合を前提に設計した．そのため既存研究では対象としていない，開発者の判断が必要とされる Dockerfile スメルの改善に効果があると期待できる．

9　おわりに

本研究では Docker イメージの品質改善を目的とした Dockerfile 最適化支援ツールを試作した．さらに，提案ツールの有効性を評価するために被験者実験を実施した．実験の結果，提案ツールはイメージサイズ削減タスクに対して正確性と効率性の向上に効果があることを確認した．一方で，ビルド時間削減タスクに対しては提案ツールが有効であるとはいえない．実験の結果と事後アンケートの回答結果を踏まえ，差分表示画面に対して改善を行った．

今後の課題として，提案ツールの処理効率の改善が挙げられる．具体的には，動作環境に依存しない実装方法の見直しや計算量を意識したアルゴリズムの検討が求められる．また，提案ツールの再評価が必要である．再度被験者実験を実施し，ビルド時間削減タスクの平均正解率と，事後アンケートの処理効率に関する回答に変化が見られるか調査すべきである．

さらに，提案ツールの発展的な改善を検討している．1 つ目は視認性の改善である．イメージ全体のサイズやビルド時間の推移を折れ線グラフで表現したり，各レイヤの棒グラフを 3 次元空間上で表現するなど，可視化表現に工夫の余地がある．2 つ目は最適化作業全体を包括的に支援する改善である．本研究では，最適化作業の 4 ステップのうち「STEP1：問題点の発見」と「STEP4：修正効果の把握」に着目していた．今後は「STEP2：問題点の分析」と「STEP3：問題点の修正」に対しても動的情報を活用し，問題点の要因と修正案の提示機能を追加すべきである．たとえばレイヤごとの依存パッケージを特定し，どのレイヤにも依存しないパッケージを提示して不要なパッケージの削除を支援する機能や，レイヤの変更頻度をもとに，コマンドの順序を最適化してビルドキャッシュの活用を支援する機能などが挙げられる．これらの改善によって提案ツールの価値の大幅な向上が期待できる．

謝辞

本研究を遂行するにあたり，多くの方々に多大なるご支援を賜りました．

楠本真二教授には中間報告会において数多くのご助言を賜りました．客観的な視点からご指摘をいただき，自分では気づけなかった貴重な改善点を得ることができました．また，研究生活中には度々菓子などの差し入れをしていただき，日々の研究作業を快適に進めることができました．3 年次に受講したソフトウェア構成論の授業を今でも覚えています．本授業からソフトウェア工学への興味が一層深まり，楠本研究室を知るきっかけになりました．楠本教授と出逢えたことに心から感謝しております．

柗本真佑准教授には研究テーマの決定から実験設計，論文や発表資料の添削に至るまで，研究のあらゆる場面で多大なるご指導を賜りました．夏季休業中で全体のミーティングがない間でも，何度も個人的なミーティングでお世話になりました．私にとって初めての研究活動でありながらここまで円滑に進められたのは，先生の温かいご支援の賜物であると確信しております．また研究のみならず，コミュニケーションの取り方や分かりやすい説明の仕方，見やすい図表や資料のデザインなど，社会に出るうえで重要な多くのスキルも学ばせていただきました．今後の仕事でも最大限に活かしてまいります．

肥後芳樹教授には中間報告会で一度ご指摘をいただく機会があり，非常に鋭いご指摘を賜りました．リファクタリングと最適化の違いといった根本的な用語から考え直すきっかけをいただき，自身の研究を一層深めることができました．また研究会出張時の夕食でご一緒し，楽しいひと時を過ごすことができました．

事務補佐員の橋本美砂子様には出張手続きや被験者実験の手続き，研究室の機材購入など研究生活を送る中で多岐にわたるサポートをしていただきました．さらに研究室内のイベントにも多くのご協力をいただき，深く感謝しております．何不自由なく快適な研究生活を送ることができたのは，橋本様のご尽力のおかげです．

楠本研究室の先輩方には研究室の習慣や研究の進め方など，多くのことを教えていただきました．論文や輪講時の発表内容についても，懇切丁寧にご助言いただき感謝の念に堪えません．また研究室旅行や新年会などのイベントを企画していただき，雑談を通じて楽しい時間を過ごすことができました．適度な休息をとりながら心地よい研究生活を送ることができました．

楠本研究室の同期の皆様には研究を進める上で分からないことを気軽に相談させていただきました．同じ目標を持つ仲間がいることが，大きな心の支えになっていました．短い間ではありましたが，感謝の気持ちでいっぱいです．

そしてここまで大学生活を無事に送ることができたのは，経済面と精神面の両面で支えてくれた家族の存在があったからこそだと実感しています．

最後に，本研究にご支援いただいた全ての方々に心より感謝申し上げます．

参考文献

[1] Muzumdar, P., Bhosale, A., Basyal, G. P. and Kurian, G.: Navigating the Docker Ecosystem: A Comprehensive Taxonomy and Survey, *Asian Journal of Research in Computer Science*, Vol. 17, No. 1, pp. 42–61 (2024).

[2] Boettiger, C.: An introduction to Docker for reproducible research, *Operating Systems Review*, Vol. 49, No. 1, pp. 71–79 (2015).

[3] Wu, Y., Zhang, Y., Xu, K., Wang, T. and Wang, H.: Understanding and Predicting Docker Build Duration: An Empirical Study of Containerized Workflow of OSS Projects, in *Proceedings of International Conference on Automated Software Engineering*, pp. 1–13 (2022).

[4] Gkamas, T., Karaiskos, V. and Kontogiannis, S.: Performance evaluation of distributed database strategies using docker as a service for industrial IoT data: Application to industry 4.0, *Journal on Information*, Vol. 13, No. 4, p. 190 (2022).

[5] Qian, J., Wang, Y., Wang, X., Zhang, P. and Wang, X.: Load balancing scheduling mechanism for OpenStack and Docker integration, *Journal on Cloud Computing*, Vol. 12, No. 1, p. 67 (2023).

[6] Rosa, G., Scalabrino, S., Bavota, G. and Oliveto, R.: What Quality Aspects Influence the Adoption of Docker Images?, *Transactions on Software Engineering and Methodology*, Vol. 32, No. 6, pp. 1–30 (2023).

[7] Rosa, G., Zappone, F., Scalabrino, S. and Oliveto, R.: Fixing dockerfile smells: An empirical study, *Journal on Empirical Software Engineering*, Vol. 29, No. 5, p. 108 (2024).

[8] Lin, C., Nadi, S. and Khazaei, H.: A Large-scale Data Set and an Empirical Study of Docker Images Hosted on Docker Hub, in *Proceedings of International Conference on Software Maintenance and Evolution*, pp. 371–381 (2020).

[9] Ksontini, E., Kessentini, M., N. Ferreira, do T. and Hassan, F.: Refactorings and Technical Debt in Docker Projects: An Empirical Study, in *Proceedings of International Conference on Automated Software Engineering*, pp. 781–791 (2021).

[10] Russell, S., Bennett, T. D. and Ghosh, D.: Software engineering principles to improve quality and performance of R software, *Journal on PeerJ Computer Science*, Vol. 5, No. 1, p. e175 (2019).

[11] Dreyfus, H. L. and Dreyfus, S. E.: A Five-Stage Model of the Mental Activities Involved in Directed Skill Acquisition, Technical Report ORC-80-2, Operations Research Center, University

of California, Berkeley (1980).

[12] Haque, M. U., Iwaya, L. H. and Babar, M. A.: Challenges in docker development: A large-scale study using stack overflow, in *Proceedings of International Symposium on Empirical Software Engineering and Measurement*, pp. 1–11 (2020).

[13] Likert, R.: A Technique for the Measurement of Attitudes, *Journal on Archives of Psychology*, Vol. 22, No. 140, pp. 5–55 (1932).

[14] Chotisarn, N., Merino, L., Zheng, X., Lonapalawong, S., Zhang, T., Xu, M. and Chen, W.: A Systematic Literature Review of Modern Software Visualization, *Journal on Visualization*, Vol. 23, No. 4, pp. 539–558 (2020).

[15] Lacerda, G., Petrillo, F. and Pimenta, M. S.: DR-Tools: a suite of lightweight open-source tools to measure and visualize Java source code, in *Proceedings of International Conference on Software Maintenance and Evolution*, pp. 802–805 (2020).

[16] 兼光智子, 肥後芳樹, 楠本真二：プログラム依存グラフを用いたリファクタリング候補の特定と可視化, 電子情報通信学会技術研究報告, pp. 61–66 (2010).

[17] Schaad, P., Ben-Nun, T. and Hoefler, T.: Boosting performance optimization with interactive data movement visualization, in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16 (2022).

[18] Romano, S., Capece, N., Erra, U., Scanniello, G. and Lanza, M.: On the use of virtual reality in software visualization: The case of the city metaphor, *Journal on Information and Software Technology*, Vol. 114, pp. 92–106 (2019).

[19] Henkel, J., Bird, C., Lahiri, S. K. and Reps, T.: Learning from, understanding, and supporting DevOps artifacts for docker, in *Proceedings of International Conference on Software Engineering*, pp. 38–49 (2020).

[20] Durieux, T.: Empirical Study of the Docker Smells Impact on the Image Size, in *Proceedings of International Conference on Software Engineering*, pp. 2568–2579 (2024).

[21] Xu, J., Wu, Y., Lu, Z. and Wang, T.: Dockerfile TF smell detection based on dynamic and static analysis methods, in *Proceedings of International Conference on Computers, Software, and Applications*, pp. 185–190 (2019).