修士学位論文

題目

Java プロジェクトと Android アプリケーションを対象としたビルド可能性調査

指導教員
肥後 芳樹 教授

報告者
小池 耀

令和 5 年 2 月 1 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

令和 4 年度 修士学位論文

Java プロジェクトと Android アプリケーションを対象としたビルド可能性調査

小池 耀

内容梗概

Gradle や Maven のようなビルドツールは開発者のビルドを容易にし，現代のソフトウェア開発で不可欠な存在となっている．しかし，ビルドツールを用いた自動的なビルドは度々失敗し，その修正に多大なコストを発生させながら，ソフトウェア開発を遅延させている．そのため，ビルドツールの実行によるビルド可能性に関する様々な研究が行われている．一方，近年普及している Android アプリケーションにおけるビルドについては，ほとんど研究が行われていない．

そこで，本研究では，Github から収集した約 7,000 件の Java プロジェクトおよび Android アプリケーションに対して自動的なビルドを実行し，ビルドの成功率や出力されたエラーを調査した．また，Android アプリケーションに対しては，出力されたエラーの原因とその解決方法について調査した．

その結果，Java プロジェクトや Android　アプリケーションの自動的なビルドの成功率は低く，大きな問題を抱えていることが分かった．また，Android アプリケーションのビルドエラーの原因として，file not found と依存関係に関するエラーが多いことが分かった．

主な用語

ビルド
Android
オープンソースソフトウェア
リポジトリマイニング

目 次

1 はじめに

ビルドは，ソースコードや各種リソースファイルを入力とし，ライブラリのダウンロード、環境に合わせて修正、コンパイル、リンク、オブジェクトの生成、配置などの一連の作業を経て実行可能なファイルを出力することである．

ソフトウェアを開発する際や，Git などで公開されているソースコードからソフトウェアを利用する際には，ビルドが必要となる．しかし，完全に手作業でソフトウェアをビルドすることは，かなりの労力を要する．そのため，Gradle や Maven のようなビルドツールが一般的に用いられている．こうしたビルドツールはビルドに必要な作業を専用のファイルに記述することで，コンパイルや依存関係の解決といった作業を自動的に行う．これにより，ビルドを一つのコマンドのみで実行することができる．

ビルドツールにより，ビルドが容易に行えるようになったことで，ソフトウェアの構築以外の目的で利用することも容易になった．一例として，多数のプロジェクトを集め，コンパイル及びコンパイルの逆の処理であるデコンパイルを通して，ソースコードを解析する研究がある [11]．このような研究では，収集した多数のプロジェクトの自動的なビルドが正しく行われることが必須である．

しかし，ビルドツールを利用したソフトウェアの多くが，自動的なビルドに失敗し，何らかの手動の操作が必要な状態にあることが先行研究で明らかになっている [7, 13, 12]．ビルドを自動的に行えず，複数の手作業による操作が必要な状況は一般的に望ましくないとされている [15]．また，ビルドを利用した研究において，十分な量のデータを得ることも困難になる．よって，ビルドが手動の変更作業などを伴うことなく，完全に自動で行えることは，重要である．

ビルドツールを用いたビルドが失敗する原因を調査した先行研究は存在する [8, 10, 12, 13]．しかし，これらの研究は特定の環境や言語を対象としており，未だ調査が行われていない環境や言語は多数存在している．

そこで，本研究ではオープンソース Android アプリケーションのビルドに焦点を当てる．Android アプリケーションは，スマートフォンの普及に伴い増加しているが，これらのビルドに関する先行研究は存在していない．そのため，オープンソース Android アプリケーションのビルドの成功率や失敗する原因などは不明であり，調査が必要である．

本研究では，Java プロジェクトに対する先行研究のスクリプトを用いて GitHub のリポジトリからフォーク数などの一定の条件を満たすオープンソース Android アプリケーションを収集し，自動的なビルドを試みる．その後得られたログから成功率やエラーの原因などをまとめた．また、出力されたエラーの原因とその解決方法について調査した．

以降，2 章では本研究の背景について述べる．3 章では，本研究での調査方法について述

べる．4 章では調査結果について，5 章では調査結果に対する考察を述べる．最後に 6 章でまとめについて述べる．

2 背景

2.1 ビルドツール

ソースコードを利用可能なソフトウェアにするためには，コンパイルや依存関係の解決など，様々な作業が必要となる．これらの作業を手作業で行うことは，かなりの労力が必要となり，操作ミスなどでソフトウェアの品質を損ねる危険も存在する．

手作業でのビルドを避けるために，ビルドで必要な作業の内コマンドのみでできる作業をスクリプトにまとめたビルドスクリプトを作成する方法が存在する．しかし，ビルドスクリプトのみでは依存関係の管理やビルド環境の指定などを行うことができない．

ビルドスクリプトによるビルドは様々な問題を抱えているため，ビルドの支援を目的とするツールであるビルドツールが存在している．ビルドツールは専用のファイルにビルドに必要な作業やデータを記入することによって，ソースコードのコンパイルや依存関係の解決，テストの実行など必要な作業を自動的に実行することができる．そのため，実際のソフトウェア開発においては，Gradle や Maven のようなビルドツールを用いてビルドを行うことが一般的である．

2.2 自動的なビルド

ビルドツールを用いたビルドは，ビルドツールを起動する一つのコマンドのみで可能である．本研究ではこのようなビルドを自動的なビルドと呼称する．

近年では，様々な場面で自動的なビルドが必要とされている．

まず，通常のソフトウェア開発において，複数のコマンドや作業が必要なビルドはエラーを誘発する [15] ため，自動的なビルドを行うことが望ましいとされている．

2.2.1 継続的インテグレーション

近年のソフトウェア開発では複数の開発者によって作成されたソースコードを統合する労力を削減するために，継続的インテグレーションと呼ばれる手法が用いられている．継続的インテグレーションでは，ソースコードの自動ビルドとテストを頻繁に行うことで，ソースコードの統合のリスクを低下させている [4]．継続的インテグレーションでは自動的なビルドを頻繁に行うため，自動的なビルドが実行可能である必要がある．

2.2.2 ビルドを用いたソースコード解析

ビルドツールを用いたビルドは，自動的に行えることから，ソフトウェアの構築以外の目的で利用することも容易になった．例えば，多数のソフトウェアを収集し，それらをコンパ

イルした後，コンパイルの逆の処理であるデコンパイルをすることで，新たなソースコードを作成し，解析する研究がある [11]．このような研究では，収集した多数のソフトウェアを正しくビルドできることが必要である．

2.3 自動的なビルドに関する先行研究

こうした自動的なビルドの現状について調査した M. Sulír らの先行研究について説明する [13]．この先行研究の目的は，様々なビルドツールが存在するにもかかわらず，頻繁にオープンソースプロジェクトのビルドが失敗しているという経験を定量的に調査することである．GitHub から 7,264 個の Java プロジェクトを収集し，自動的なビルドが成功するか否かを調査した．その結果，38%以上のプロジェクトがビルドに失敗した．その主要な要因として依存関係に関するエラーがあることを示した．また，大規模なプロジェクト，古いプロジェクト，更新が古いプロジェクトは，ビルドが失敗する可能性が高いことを示している．

上記の研究は 2016 年に行われたが，2020 年にも同様の調査が行われた [12]．その結果，59.4%のプロジェクトがビルドに失敗しており，ビルドがより失敗しやすい状況になっていることが分かった．

一方，先行研究からビルドツールを用いた自動的なビルドの現状が示されたが，Java という特定の言語のみを対象としており，いまだに調査が行われていない環境や言語は多数存在している．また，先行研究では，2016 年の研究は Java8 を用いて行われているが，2020 年の研究では Java11 を用いている．Java 8 と Java11 には非互換性があり，この影響については調査がなされていない．

2.4 Android アプリケーション

本研究ではオープンソース Android アプリケーションのビルドに焦点を当てる．Android アプリケーションは，スマートフォンのうち，72.37%でオペレーティングシステムとして利用されている AndroidOS 上で動作するアプリケーションである．AndroidOS が広範に利用されていることから，Android アプリケーションは現代社会において重要な役割を果たしているといえる．一方で，これらのビルドに関する先行研究はほとんど存在せず [3, **?**]，Android アプリケーションの自動的なビルドの現状を調査した先行研究は存在しない，

そのため，Android アプリケーションのビルドの成功率や失敗原因といった情報は不明であり，調査が必要である．

2.5 ビルドに関するデータセット

ビルドに関する研究は様々なものがあるが，その中に自動的にビルドを修正する方法に関する研究がある [5, 6, 9, 14].

Hassan らは，は GitHub の上位 200 個の Java プロジェクトに対して自動的なビルドを実行し，失敗した 86 個の Java プロジェクトのビルドの実行方法を手動で調査した．その結果少なくとも 57%のビルド失敗を自動的に解決できることを示した [5]．また，TravisTorrent というデータセットから，175 件のビルド失敗とビルドの修正方法を抽出した．そして，そのうち 135 件を用いて修正パターンを生成し，残りの 40 件に適応した．その結果再現性のあるビルド失敗 24 件のうち 11 件については手動修正と同程度の時間で修正できることを明らかにした [6].

Macho らは，23 個の Java プロジェクトにおける 37 個の失敗した Maven ビルドを手動で調査し，バージョン更新、依存関係の削除、およびリポジトリの追加という 3 つの修正方法が有効であることを発見した．そしてこれらの方法の有効性を，23 個の Java プロジェクトから 84 個の失敗したビルドを追加で抽出し適用することで評価した．結果 54%の失敗したビルドを自動で修復できることを明らかにした [9].

Vassallo らは，Maven のビルドが失敗する原因をまとめ，インターネット上から収集した解決方法を提案する BART というツールを作成した．このツールによりビルドの修正時間が平均 37%短縮された．

このような研究では，ビルドに関するデータセットが必要になる．このようなデータセットは存在しているが [2]，いずれも継続的インテグレーションを用いている．継続的インテグレーションを用いていない Android アプリケーションを含む，大規模な Android アプリケーションのビルドから構築したデータセットは存在しない．

3 調査方法

本研究ではまず，Java プロジェクトに対して先行研究と同様の調査を実行する．その後オープンソース Android アプリケーションのビルドの成功率や失敗原因について調査する．

以下の Research Question を設定した．

- RQ1: 現在の Java プロジェクトや Android アプリケーションにおけるビルド成功率はどの程度か．また，その他のデータはどのようになっているか
- RQ2: Java 8 と Java 11 の非互換性の影響はどの程度か
- RQ3: Android アプリのビルドの失敗原因は何かまたそれを解消するにはどうすればよいか

RQ1 では現在の Java プロジェクトや Android アプリケーションにおけるビルドの成功率を調査することで，これらのビルドに問題が存在するか否か明らかにする．また，ファイル数や最終更新日などのデータとビルドの関係などを調査し，自動ビルトの現状を定量的、定性的に分析する．

RQ2 では先行研究では調査されていない Java 8 と Java11 の非互換性が自動的なビルドに与えている影響を調査する．

RQ3 では，Android アプリケーションのビルドの失敗原因や対処方法を調査し，Android アプリケーションのビルドを改善させる方法を明らかにする．

これらの Research Question に回答するために，まず，Java プロジェクトに対して先行研究と同様の調査を実行し，2016 年と 2020 年の結果と比較する．その後，Java プロジェクトに対する先行研究の手法を用いて，GitHub から一定の条件を満たすオープンソース Android アプリケーションを収集し，ビルドを実行する．その後，得られたログからビルドの成否やビルドエラーをまとめたデータセットを構築する．また，エラーをその原因によって手動で分類し，それぞれ対処方法について調査する．

3.1 調査対象

本研究では，先行研究と同様の条件で GitHub から Java プロジェクトを抽出し調査対象とした．その条件は以下の通りである．

1. Java で記述されているアプリケーションのリポジトリであること
2. オープンソースソフトウェアのライセンスが記述されたファイルがあること
3. 1 回以上フォークされていること

4. JNI， Java Mava， Android の API を用いていないこと

これらの基準が設けられている理由について説明する．

条件（1）については，Java は Gradle， Maven， Ant などの高機能なビルドツールが複数存在し，Android アプリケーションにおいても頻繁に用いられるため，調査対象として選択した．

条件（2）については，オープンソースのソフトウェアを対象にしているため，そのソフトウェアが用いているライセンスについて記述されたファイルが存在していることを条件として設定した．

条件（3）は，個人的なリポジトリを排除し，他者からの協力を受けたリポジトリを対象とするために設定した．

そして， 条件（4）は，JNI や Java ME，Android といった技術を排除し，の使用を確認することで， 純粋な Java プロジェクトに焦点を当て，他の言語に関するデータが入らないようにするために設定した．

そして，以下の条件を満たすリポジトリをオープンソース Android アプリケーションとして調査対象とした．

1. Kotlin で記述されているアプリケーションのリポジトリであること
2. オープンソースソフトウェアのライセンスが記述されたファイルがあること
3. 1 回以上フォークされていること
4. JNI や Java Mava の API を用いていないこと
5. Android の API を用いていること

条件（2）から（4）までは同様の理由のため，条件（1）と条件（5）について説明する．条件（1）で Kotlin を選択したのは，Kotlin が Android 開発における推奨言語であり，Java から派生した言語であることから，調査対象として適切だと考えたためである．そして，条件（5）で Android の API を用いていることを条件にすることで，オープンソース Android アプリケーションを抽出できるようにしている．

これらの条件を満たす Java プロジェクト，Android アプリケーションをそれぞれ 10,000 個 GitHub から抽出し，調査対象とした．

3.2 ビルドの手順

この章では，上記の条件を満たすリポジトリに対して行うビルドの手順について説明する，

3.2.1 ビルドツールの選択

Java の主なビルドツールとして，Gradle， Maven， Ant の 3 つが存在する．ルートディレクトリ内にこれらのシステムが用いるビルドファイルが存在しているか確認した．もし複数のビルドファイルが存在している場合は，Gradle， Maven， Ant の順に優先する．これは，Gradle， Maven， Ant の順に新しいビルドツールであり，新しいビルドツールが主に使用され，古いビルドツールは互換性のために削除していないという状況を想定しているためである．ただし，オープンソース Android アプリケーションの調査については，Gradle のみを用いる．これは，Android アプリケーションは開発環境の影響でほぼ 100%が Gradle を用いているためである．また，オープンソース Android アプリケーションのビルドには Gradle wrapper を用いる．Gradle wrapper は Gradle をインストールしていない環境や，異なるバージョンの Gradle を用いた環境でも，開発者が指定した Gradle のバージョンでビルドを行える Gradle の機能である．Android アプリケーションの調査において Gradle wrapper を用いるのは，一般的な Gradle のビルドにおいては Gradle wrapper が用いられていると考えられるためである．

調査対象のルートディレクトリ内にこれらのシステムが用いるビルドファイルが存在しているか確認した．結果として Java プロジェクトは 7,196 個，Android アプリケーションは 7,362 個にビルドファイルが確認された．これらの Java プロジェクト，Android アプリケーションに対してビルドを行う．

3.2.2 ビルド環境

本研究では，ビルドを Docker 環境で行った．これは， 調査対象の言語の開発者の環境を高い再現性を持ってシミュレーションするためである．Java プロジェクトのビルドに用いる Docker のイメージは先行研究で使用されたものを一部修正して活用した．先行研究で用いられたイメージは以下のようなソフトウェアで構成されている．

1. RPM 系 Linux ディストリビューション Fedora
2. JDK 8
3. Gradle 6.5.1 ， Maven， Ant
4. Apache Ivy
5. Git
6. Ruby

また，オープンソース Android アプリケーションのビルドには，mingc/android-build-box という Android アプリケーションのビルドのために作成された Docker コンテナを用いた [1]．このコンテナに，調査に必要なソフトウェアを追加し以下のようなソフトウェアで構成した．

1. Ubuntu 18.04
2. JDK 8 ，JDK 11
3. Gradle 6.5.1
4. Android SDK 25 から 30
5. Android NDK r21
6. Git
7. Ruby

3.3 ビルドツールの制御

調査の制御を行う制御スクリプトの動作について説明する．まず，Java プロジェクトに対する制御スクリプトの動作について説明する．制御スクリプトは GitHub から調査対象の条件を満たすリポジトリからソースコードをコピーする．その後各リポジトリのビルドファイルに基づいたビルドツールに対応するコマンドを実行する．使用したコマンドを表 1 に示す．

仕様ツール	ビルドファイル名	コマンド
Gradle	Build.gradle	gradle clean assemble
Maven	pom.xml	mvn clean package -DskipTests
Ant	build.xml	ant clean; ant jar || ant war || ant dist || ant

表 1: 使用したコマンド

各ビルドでは，ビルドツールが正常に動作したか失敗したかを確認するため，標準出力とエラー出力をログファイルに出力する．また，ビルドの際に生成したアーカイブファイルが正しく動作するかテストを実行する場合があるが，テストに失敗した場合でも，アーカイブファイルが出力されていれば，ソフトウェアが正常に動作する可能性が高い．そのため，テストをスキップできるビルドツールではスキップしている．

3.4 収集するデータ

調査対象や調査対象のビルド結果から収集するデータについて説明する．まず，リポジトリから得られるデータとして，以下のデータがある．

- リポジトリ名
- 使用しているビルドツール
- スター数
- フォーク数
- リポジトリが作成された日時
- リポジトリが最後に更新された日時
- ウォッチャー数
- コミット数
- イシュー数
- ビルドに関連するツールのファイルの有無 (Travis CI で用いる travis.yml の有無など)
- ファイル数

これに加え，ビルド後に以下のデータを収集する．

- ビルドの成否
- 終了コード
- ビルドにかかった時間
- ビルド終了後のファイル数
- ビルドログ

これらのデータを収集し，csv ファイルに記述し，データセットを構築した．

3.5 ビルドエラーの抽出と分類

この章では，ログからビルドエラーを抽出する方法とビルドエラーをその原因によって分類する方法について説明する．

最初に，ログを検索し，特定の位置に存在する文字列を抽出することで，ビルドエラーを抽出する．ログの形式はビルドツールによって異なるため，ビルドツール毎に対応した抽出方法が必要である．それぞれのビルドツール毎の抽出方法について簡単に説明する．

3.5.1 Ant

Ant は全てのビルドエラーに対して BuildException というエラーを表示するため，ビルドエラーそのものの抽出は行えない．しかし，Ant は実行時にターゲットと呼ばれるビルドに必要なタスクの集合を表示する．最後に表示されたターゲットが失敗したターゲットと言えるため，これを検索し抽出している．

3.5.2 Gradle

Gradle のビルドエラーでは，一つの例外が複数の例外を原因として発生することで，例外がツリー構造で形成されることがある．例として，失敗したビルドのログから失敗原因を抜き出し簡略化したものを以下に示す．

```
* Exception is:TaskExecutionException:
Caused by: DefaultLenientConfiguration
$ArtifactResolveException:
Cause 1: ModuleVersionResolveException:
Caused by: ModuleVersionResolveException: 中略
Cause 2: ModuleVersionResolveException: 中略
```

上記の例では,「TaskExecutionException」が「DefaultLenientConfiguration $ArtifactResolveException:」によって引き起こされ，さらに，そのエラーが「Cause 1: ModuleVersionResolveException」と「Cause 2: ModuleVersionResolveException」によって引き起こされている．これらのエラーは同じ名前だが，異なった原因によって引き起こされている.「Cause 1: ModuleVersionResolveException」はさらに異なる「ModuleVersionResolveException」によって引き起こされている．このように，一つの例外が複数の例外が原因で引き起こされる場合がある．

一つのビルドに複数のエラータイプを割り当てると，その後の解析が複雑になるため，最も関連性の高い例外を以下の方法で決定する．例外がツリーを形成しない単一の例外であれば，それを選択する．例外がツリー構造になっている場合は，根となる例外を引き起こしている例外を選択する．上記の例の場合は，根となる例外の「TaskExecutionException」を引き起こしている,「DefaultLenientConfiguration $ArtifactResolveException」を選択する．

3.5.3 Maven

Maven のビルドエラーにおいても複数の例外が表示される場合がある．Maven では発生した例外に関連する Web ページの URL が表示されるため，この URL から例外を起こした

クラス名を抽出している．URL が複数含まれている場合は，最初の URL からクラス名を抽出した．これは最初の URL に最も具体的な例外が含まれている場合が多いためである．

3.5.4 ビルドエラーの分類

上記の方法でビルドエラーを抽出した結果，Java プロジェクトからは 502 種類，Android アプリケーションからは 154 種類のエラーが発見された．これらのエラーは非常に数が多く，人間には読みにくいため，より分かりやすいように先行研究の方法を参考にしてビルドエラーが起きた原因によって分類を行った．これらの作業は手作業のため，頻繁に発生するビルドエラーのみを対象としている．結果として Java プロジェクトからは 84 種類，Android アプリケーションからは 54 種類のエラーを分類した．なお，手動の分類の際，原因が分からなかったものは未分類のままとした．それぞれの分類の説明を表 2 に示す．

なお，Android アプリケーションについては一つの調査対象に対し，Java 8 と Java 11 それぞれについてビルドを行うため，二種類のビルドエラーが抽出され，二種類の分類結果が得られる， しかしほとんどの場合は，どちらも同じエラーか，片方が間違った JDK のバージョンによるものであり，双方が間違った JDK のバージョン以外の分類結果でなおかつ双方が異なった分類結果になる場合は全体の約 0.36%であった．

そのため，双方が異なるビルドエラーだった場合，片方が間違った JDK のバージョンによるものだったときはもう片方のビルドエラーを用いて分類を行う．もう片方の原因が分からず，未分類となった場合は，間違った JDK のバージョンに分類する．双方ともに間違った JDK のバージョン以外の分類結果だった場合は，Java 8 の分類結果を用いた．

分類名	説明
ビルドファイルの構文エラー	ビルドファイルの構文によるエラー
設定	ビルドの設定によるエラー
依存関係	依存関係に関するエラー
ドキュメント生成	javadoc などのドキュメント生成に関するエラー
外部プログラムの実行	他のプログラムやプロセスとの通信時のエラー
File not found	ファイルが見つからないために発生するエラー
Java コンパイル	Java のソースコードのコンパイル時に発生するエラー
Kotlin コンパイル	Kotlin のソースコードのコンパイル時に発生するエラー
他の言語のコンパイル	Java や Kotlin 以外のソースコードのコンパイル時に発生するエラー
他のビルドツールとの統合	他のビルドツールとの統合時に発生するエラー
パッケージング	成果物の出力時のエラー
バージョン管理システム	Git などのバージョン管理システムに関するエラー
間違った JDK のバージョン	Java のバージョンが誤っているために発生したエラー

表 2: 分類の説明

4 調査結果

本章では，Java プロジェクトと Android アプリケーションのビルドの状況について調査した結果をまとめる．

4.1 RQ1: 現在の Java プロジェクトや Android アプリケーションにおけるビルド成功率はどの程度か，また，その他のデータはどのようになっているか

4.1.1 ビルドの成功率

2016 年と 2020 年の先行研究における Java プロジェクトのビルド成功率と 2022 年現在の Java プロジェクトと Android アプリケーションのビルド成功率を表 3 にまとめる．2016 年の結果では成功率が 61.87%となっているがほかの結果は全て 40%程度となっている．

4.1.2 リポジトリ作成日や最終更新日とビルドの成否の関係

2016 年の Java プロジェクトのリポジトリ作成日と最終更新日からの日数とビルド成功率との関係をバイオリンプロットで図 1 に示す．図 1 の左図の縦軸はリポジトリ作成からの日数、横軸は成功／失敗したプロジェクトの密度を示しており，実線は中央値を示している．

言語	成功率	失敗率	タイムアウト率
Java プロジェクト 2016 年	61.87%	38.05%	0.08%
Java プロジェクト 2020 年	40.49%	59.38%	0.12%
Java プロジェクト（Java 8） 2022 年	46.1%	53.75%	0.15%
Java プロジェクト（Java 11） 2022 年	43.02%	56.6%	0.38%
Android アプリ（Java 8） 2022 年	40.97%	58.95%	0.08%
Android アプリ（Java 11） 2022 年	48.11%	51.83%	0.05%

表 3: ビルド成功率の変化 (太字は本研究結果)

また図 1 の右図の縦軸は最終更新日からの日数、横軸は成功／失敗したプロジェクトの密度を示している．図 1 は失敗したビルドのほうがファイル数の中央値が大きいため，古く，最後の更新が遅い Java プロジェクトのほうが，失敗しやすい傾向にあることを示している．

2020 年と 2022 年の Java プロジェクトと Android アプリケーションにおける同様の図をそれぞれ図 2，図 3，図 4，図 5，図 6 に示す．図 2，図 3，図 6 は，図 1 と同様に，古く，更新が遅い Java プロジェクトや Android アプリケーションが失敗しやすい傾向を示している．

一方，図 4 はやや新しく，最近更新された Android アプリケーションが失敗しやすい傾向を示している．図 5 は約 500 日以内に作成，更新されたリポジトリのビルドが失敗しやすいことを示している．中央値も成功したビルドのほうが大きく，新しく，最近更新された Android アプリケーションが失敗しやすい傾向を示している．

4.2 RQ2:Java 8 と Java 11 の非互換性の影響はどの程度か

Java プロジェクトと Android アプリケーションそれぞれについて，Java8 でのビルドの成否と Java11 でのビルドの成否を組み合わせた割合を図 7 に示す．Java プロジェクトにおいては，約 20%ほどが，Java 8 か Java11 のどちらか片方でしかビルドが成功していないことが分かる．また，Android アプリにおいては，40%弱ほどが，Java8 あるいは Java11 でしかビルドが成功していないことが分かる．

4.3 RQ3: Android アプリケーションのビルドの失敗原因は何か．またそれを解消するにはどうすればよいか

Android アプリケーションのビルドエラーを手動で分類した結果について表 7 に示す．割合として file not found が最も多く次いで依存関係のエラーが多かった．

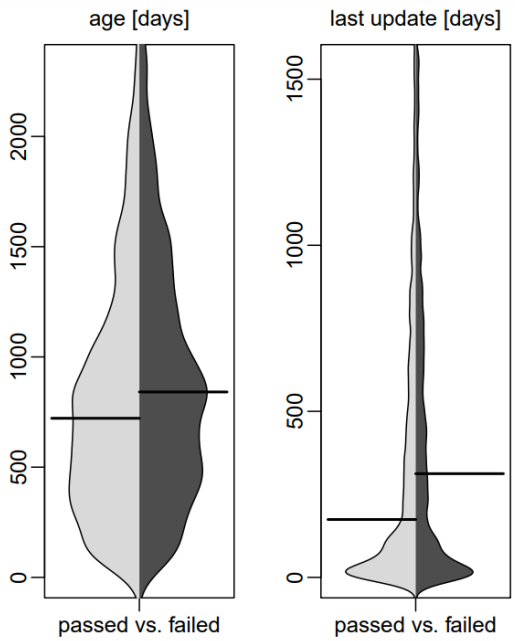

図 1: 2016 年の Java プロジェクトのリポジトリ作成日と最終更新日からの日数とビルド成功率との関係 (横線は中央値)

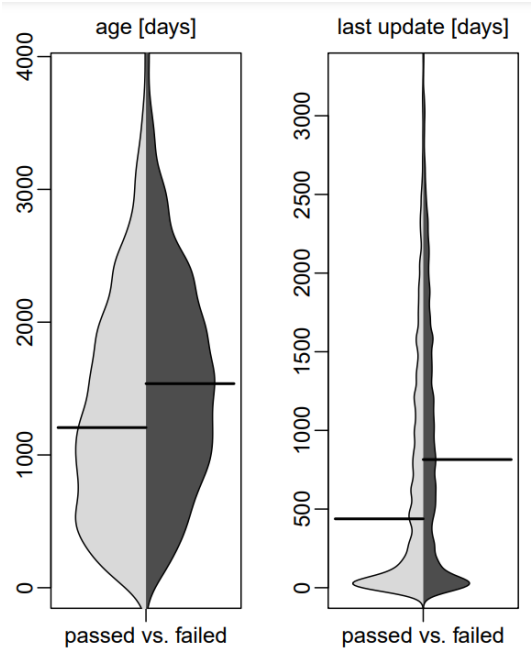

図 2: 2020 年の Java プロジェクトのリポジトリ作成日と最終更新日からの日数とビルド成功率との関係

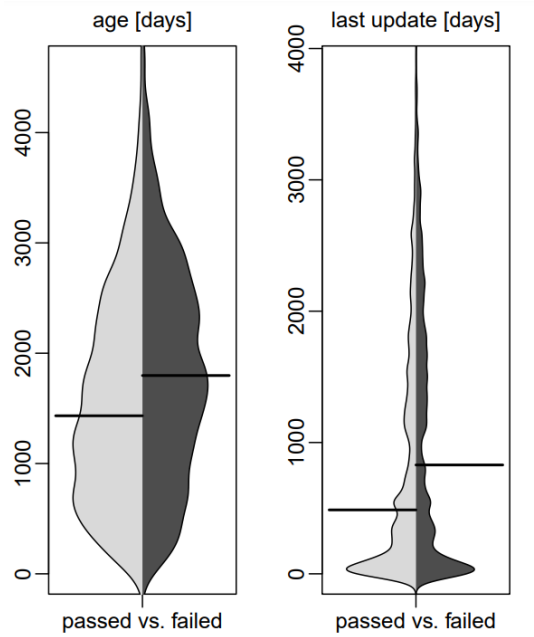

図 3: 2022 年の Java プロジェクトのリポジトリ作成日と最終更新日からの日数と Java 8 を用いたビルド成功率との関係

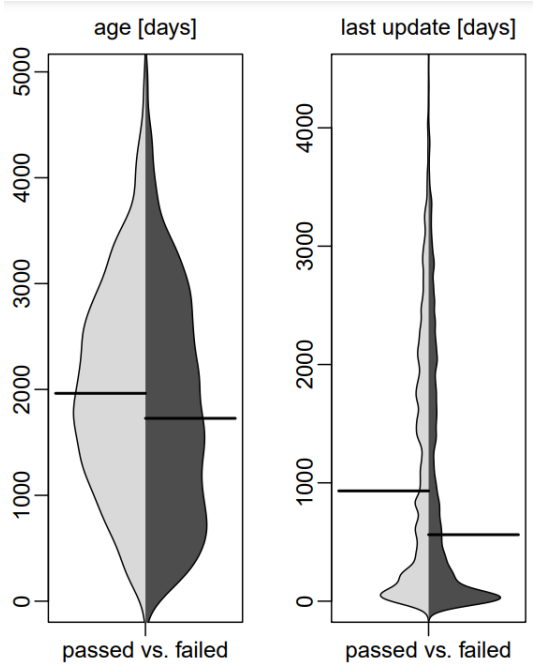

図 4: 2022 年の Java プロジェクトのリポジトリ作成日と最終更新日からの日数と Java 11 を用いたビルド成功率との関係

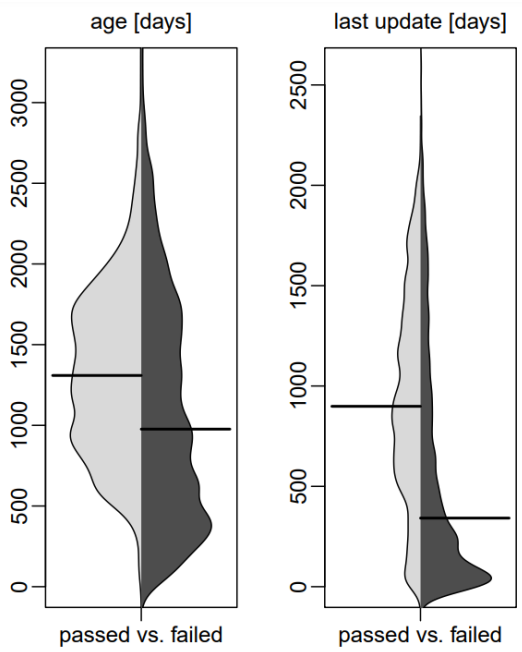

図 5: Android アプリケーションのリポジトリ作成日と最終更新日からの日数と Java 8 を用いたビルド成功率との関係

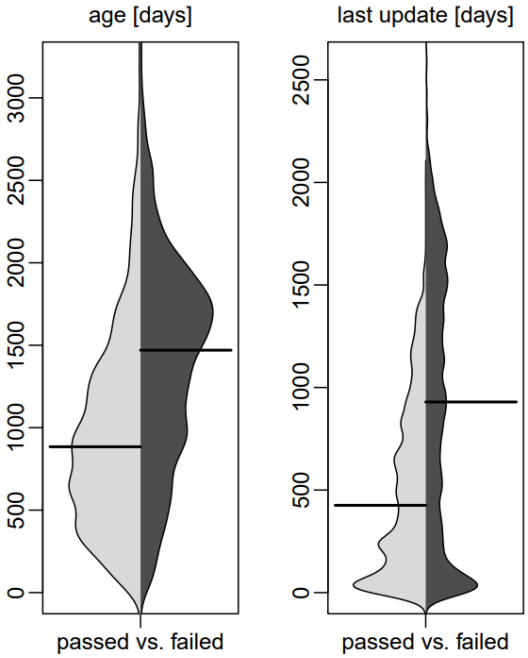

図 6: Android アプリケーションのリポジトリ作成日と最終更新日からの日数と Java11 を用いたビルド成功率との関係

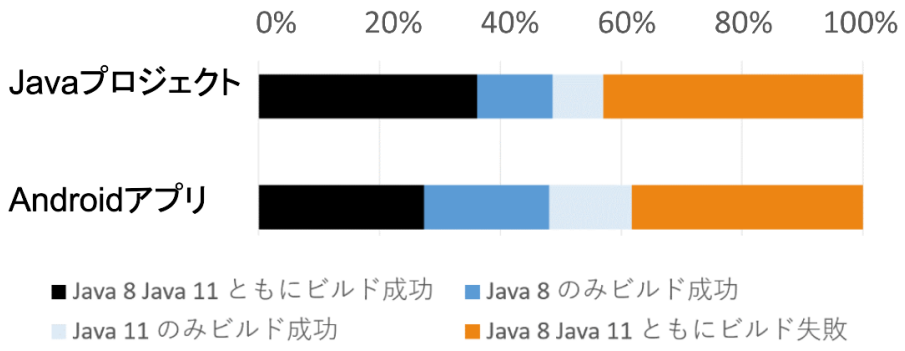

図 7: Java プロジェクトと Android アプリケーションの Java8 でのビルドの成否と Java11 でのビルドの成否の割合

5 考察

5.1 RQ1

5.1.1 ビルドの成功率

RQ1 では，2016 年の結果以外は約 40%程度の成功率となっていることが分かる．このことから，近年の Java プロジェクトおよび Android アプリケーションの多くでは自動的なビルドがうまく機能していないことが分かる．

5.1.2 リポジトリ作成日や最終更新日とビルドの成否の関係

図 1，図 2，図 3，図 6 については先行研究と同様に古く，更新が遅いリポジトリが失敗しやすいという結果だったが，図 4，図 5 は他の調査結果とは異なり，新しく，最近更新がなされたリポジトリのほうが失敗しやすい傾向を示した．

図 4 において，Java 11 を用いた Java プロジェクトのビルドが新しく，最近更新がなされたリポジトリのほうが失敗しやすい傾向を示した理由を調査した．Java 11 を用いた Java プロジェクトのビルドにおけるビルドエラーの割合上位五種を Ant，Maven，Gradle 毎にまとめたものを表 5 に，集計の対象を 2020 年以降に作成された 1,799 個の Java プロジェクトのみに変更したものを表 6 に示す．表 5 と表 6 を比較する．表 5 では，Maven のビルドエラーの 11.5%を「MojoFailure:maven-compiler-plugin」という Maven のビルドエラーが占めているが，表 6 では 22.5%に増加している．

同様に，Gradle の「PluginApplication:fabric-loom」というビルドエラーは 8%から 20.6%に変化している．このようなビルドエラーの影響で，新しく最近更新がなされた Java プロジェクトが失敗しやすくなっている可能性がある．

分類名	割合
File not found	27.84%
依存関係	19.03%
間違った JDK のバージョン	12.59%
設定	7.57%
ビルドファイルの構文エラー	6.51%
Kotlin コンパイル	6.33%
パッケージング	1.78%
外部プログラムの実行	1.42%
他の言語のコンパイル	0.53%
ドキュメント生成	0.04%
バージョン管理システム	0.04%

表 4: Android アプリケーションのビルドエラーの分類結果

Ant	Maven	Gradle
-do-compile 29.1%	MojoFailure:maven-compiler-plugin 32.6%	NoSuchMethod 10.8%
compile 20.5/%	DependencyResolution 25.1%	ArtifactResolve 9.4%
CannotImport 10.7%	MojoExecution:maven-compiler-plugin 11.5%	CompilationFailed 9%
-init-taskdefs 2.6%	UnresolvableModel 8%	PluginApplication:fabric-loom 8%
build-project 2.6%	MojoExecution:maven-enforcer-plugin 2.6%	PluginApplication:gradle 5.7%

表 5: Java プロジェクトに対する Java 11 を用いたビルドにおけるビルドエラーの割合上位五種

図 5 において，このような傾向が表れた理由を調査したところ，Android Gradle プラグインというツールが 7.0.0-alpha02 バージョンから Java 11 以上を必須条件にしたことが原因だと分かった．最近の Android アプリケーションは Android Gradle プラグインを 7.0.0-alpha02 以上のバージョンで用いているため，ビルドの成功する割合が著しく低下したと考えられる．

5.2 RQ2

図 7 から，Java プロジェクトと Android アプリケーションどちらも Java8 と Java11 の非互換性の影響を受けていることが分かったが，Android アプリケーションのほうが強い影響を

Ant	Maven	Gradle
-do-compile 34.8%	MojoExecution:maven-compiler-plugin 25.2%	PluginApplication:fabric-loom 20.6%
CannotImport 13%	MojoFailure:maven-compiler-plugin 22.5%	PluginApplication:gradle 10.4%
compile 13%	DependencyResolution 18.5%	ArtifactResolve 7.7%
-do-init 8.7%	UnresolvableModel 5.8%	UnknownPlugin 7.7%
-init-taskdefs 4.3%	MojoExecution:maven-enforcer-plugin 4%	PluginApplication:boot 6.7%

表 6: 2020 年以降に作成された Java プロジェクトに対する Java 11 を用いたビルドにおけるビルドエラーの割合上位五種

受けていることが分かった．この理由として RQ1 の考察でも触れた Android Gradle プラグインが 7.0.0-alpha02 バージョンから Java 11 以上を必須条件としていることが挙げられる．

また，Gradle の 4.2 以下のバージョンでは，Java 9 以上を利用できない点も挙げられる．これは Java9 から Java のバージョン番号の表記方法が変更されたことが原因である．

このように，Android アプリケーションでは，Java のバージョンによる制約が多いため，より強く Java 8 と Java11 の非互換性の影響を受けたと考えられる．

5.3 RQ3

RQ3 において，Android アプリケーションのビルドエラーは 10 種類に分類された．これらの原因と対処方法について分析を行った．

5.4 file not found の原因と対処方法

file not found はファイルが見つからないという一般的なエラーである．そのため原因や対処方法は様々であるが，特に割合が多かったビルドエラーに対して調査を行った．

5.4.1 Classnotfound の原因と対処方法

Classnotfound の例をソースコード 1 に示す．file not found の最も大きい割合を占めるビルドエラーは「Classnotfound」であり，ビルドエラー全体の約 6%を占めていた．また，「Classnotfound」の発生原因のほとんどは gralde-wrapper.jar というファイルが見つからないことであった．これは開発者がソースコードを git に push する際，gitignor の設定ミスにより gralde-wrapper.jar ファイルが push されないことが一因と考えられる．gradle　wrap というコマンドにより，gradle-wrapper.jar を作り直すことで対処することが可能であった．

Listing 1: Classnotfound の例

```
1: Error: Could not find or load main class org.gradle.wrapper.GradleWrapperMain
2: Caused by: java.lang.ClassNotFoundException: org.gradle.wrapper.GradleWrapperMain
```

Listing 2: Gradle:processDebugGoogleServices の例

```
 27: * What went wrong:
28: Execution failed for task ':app:processDebugGoogleServices'.
29: > File google-services.json is missing.
      The Google Services Plugin cannot function without it.
30:   Searched Location:
31:   /tmp/build/Capstone-2022-C22-PS080_sskin-mobile/app/src/debug/google-services.json
32:   /tmp/build/Capstone-2022-C22-PS080_sskin-mobile/app/src/google-services.json
33:   /tmp/build/Capstone-2022-C22-PS080_sskin-mobile/app/src/Debug/google-services.json
34:   /tmp/build/Capstone-2022-C22-PS080_sskin-mobile/app/google-services.json


 @<lstlisting>$|latex|\linebreak\hspace*{5ex}$
```

5.4.2 Gradle:processDebugGoogleServices の原因と対処方法

Gradle:processDebugGoogleServices の例をソースコード 2 に示す.「Gradle:processDebugGoogleServices」が file not found で二番目に大きい割合を占めるビルドエラーであり，ビルドエラー全体の約 5.4%を占めていた．このビルドエラーの原因は google-services.json というファイルが存在しないことである．このファイルは，Firebase という Google のモバイル プラットフォームのサービスを利用する際に必要となる．このファイルを firebase のホームページから作成し，適切な場所に配置することによってビルドエラーを解決することができた.

5.5 依存関係の原因と対処方法

依存関係の例をソースコード 3 に示す．依存関係のエラーは依存先の消滅やバージョンの互換性の問題など様々な原因により，依存関係の解決ができないことが原因である．依存関係の問題に対処するためには，ビルドの依存関係が記述されている build.gradle ファイルを修正する必要がある．例えば依存関係が原因でビルドが失敗した udex-app-android というリポジトリでは，依存先として GitHub のリポジトリのある特定のコミットを指定していた．しかし，このコミットがマージが原因で消滅したことにより，依存先がなくなり，ビルドが失敗した．そのため，同時期のコミットを新たに指定することによって解決した.

Listing 3: 依存関係のエラーの例

```
09: * What went wrong:
10: Could not determine the dependencies of task
      ':app:compileDevTestnetReleaseJavaWithJavac'.
11: > Could not resolve all task dependencies for configuration
      ':app:devTestnetReleaseCompileClasspath'.
12:    > Could not find com.github.horizontalsystems:hd-wallet-kit-android:a3666d8.
13:      Required by:
14:          project :app
```

Listing 4: 設定の例

```
302: * What went wrong:
303: Execution failed for task ':app:validateSigningRelease'.
304: > Keystore file '/tmp/build/AChep_acpods/app/acpods-release.keystore'
       not found for signing config 'release'.
```

5.6 設定の原因と対処方法

設定の例をソースコード 4 に示す．設定に関するエラーの内最も多かったビルドエラーは InvalidUserData であり，ビルドエラー全体の約 2%を占めていた．このビルドエラーの原因は Android アプリケーションを Google Play ストアに公開するために必要な署名が存在しないことが原因であった．keytool コマンドにより適当な署名を作成することで対処ができた．また，署名に関するタスクを削除することによっても対処することができた．

5.7 Kotlin コンパイルの原因と対処方法

Kotlin コンパイルの例をソースコード 5 に示す．Kotlin コンパイルのエラーは，Kotlin のソースコードに問題があり，コンパイルがうまくいかないために発生するエラーである．Kotlin　コンパイルに分類されたビルドエラーとして Gradle:compileDebugKotlin がある．このビルドエラーを起こしたリポジトリを一つ選び調査したところ，新機能の実装が中途半端に行われたことでビルドエラーを起こしていた．その部分を削除することでビルドが成功するようになった．

5.8 パッケージングの原因と対処方法

パッケージングの例をソースコード 6 に示す．パッケージングはビルドの成果物を出力する際のエラーである．パッケージングとして分類されたエラーは全て「Gradle:lintVitalRelease」であった，このビルドエラーは，Kotlin の静的解析ツールである ktlint が出力するエラーで

Listing 5: Kotlin コンパイルの例

```
66: > Task :app:compileDebugKotlin FAILED
67: e: /tmp/build/Fueled_reclaim/app/src/main/kotlin/com/fueled
      /reclaim/samples/items/LoadingAdapterItem.kt: (10, 1):
      Class 'LoadingAdapterItem' is not abstract and does not
      implement abstract base class member public abstract fun
      onCreateViewHolder(view: View):
      LoadingViewHolder defined in com.fueled.reclaim.AdapterItem
68: e: /tmp/build/Fueled_reclaim/app/src/main/kotlin/com/fueled/
      reclaim/samples/items/LoadingAdapterItem.kt: (14, 5):
      'onCreateViewHolder' overrides nothing
69: e: /tmp/build/Fueled_reclaim/app/src/main/kotlin/com/fueled/
      reclaim/samples/items/LoadingAdapterItem.kt: (14, 43):
      Unresolved reference: View
70: e: /tmp/build/Fueled_reclaim/app/src/main/kotlin/com/fueled/
reclaim/samples/items/LoadingAdapterItem.kt: (18, 31): Unresolved reference: View
71:
72: FAILURE: Build failed with an exception.
73:
74: * What went wrong:
75: Execution failed for task ':app:compileDebugKotlin'.
76: > Compilation error. See log for more details
```

ある．対処方法として，ソースコードにも記述されているように，エラーが出力された場合でもビルドを続行するよう設定する方法がある．

5.9 外部プログラムの実行の原因と対処方法

外部プログラムの実行の例をソースコード 7 に示す．外部プログラムの実行のエラーは，外部コマンドの実行が失敗した際のエラーである．外部プログラムの実行のエラーに分類されたビルドエラーの中で最も多かったのは ExecException であった．ExecException は 51 件確認された．失敗した外部プロセスを表にまとめる．

失敗した外部プログラムとして最も多かったのは git だった．しかし，git の実行の失敗によるビルド失敗は手動で再現を行うことができなかった．何らかの一時的な不具合によって git の実行が失敗したと考えられる．

5.10 他の言語のコンパイルの原因と対処方法

他の言語のコンパイルの例をソースコード 8 に示す．他の言語のコンパイルは Java や Kotlin 以外のコンパイル時のエラーである，他の言語のコンパイルに分類された Android アプリ

Listing 6: パッケージングの例

```
143: > Task :app:lintVitalRelease FAILED
144:
145: FAILURE: Build failed with an exception.
146:
147: * What went wrong:
148: Execution failed for task ':app:lintVitalRelease'.
149: > Lint found fatal errors while assembling a release target.
150:
151:   To proceed, either fix the issues identified by lint,
     or modify your build script as follows:
152:   ...
153:   android {
154:       lintOptions {
155:           checkReleaseBuilds false
156:           // Or, if you prefer, you can continue
     to check for errors in release builds,
157:           // but continue the build even when errors are found:
158:           abortOnError false
159:       }
160:   }
161:   ...
162:
```

ケーションは 15 件だった．これらの Android アプリケーションを調査したところ，画像の最適化に失敗していたのが 7 件，compileSdkVersion というモジュールが原因だったものが 7 件，その他が 1 件であった．画像の最適化に失敗しているものは，画像の最適化を設定から無効にすることで解決した．また，compileSdkVersion のエラーは，SDK のバージョンが低いことが原因であった．

5.11 ドキュメント生成の原因と対処方法

ドキュメント生成の例をソースコード 9 に示す．ドキュメント生成は javadoc を用いてドキュメントを生成する際に発生するエラーである．本研究では 1 件のみ確認された．Java11 ではなく Java8 を用いた場合には確認されず，Java11 ではなく Java8 を用いることで解決したとする Github issue の議論が存在した．そのため，Java 8 を用いることで回避できると考えられる．

Listing 7: 外部プログラムの実行の例

```
133: FAILURE: Build failed with an exception.
134:
135: * Where:
136: Build file '/tmp/build/paletteOvO_MimikkoUi-Mod/app/build.gradle' line: 7
137:
138: * What went wrong:
139: A problem occurred evaluating project ':app'.
140: > Process 'command 'git'' finished with non-zero exit value 128
141:
142: * Try:
143: Run with --info or --debug option to get more log output.
     Run with --scan to get full insights.
```

失敗した外部プログラム	個数
git	40
java	2
python	2
gradle	2
npm	1
合計	51

表 7: 失敗した外部プログラムの個数

5.12 バージョン管理システム

バージョン管理システムの例をソースコード 10 に示す．バージョン管理システムは git に関するエラーである．本研究では 1 件のみ確認されたが，原因は.git ファイルが存在しないことであり，手動での再現はできなかった．実験の際に何らかの理由で.git ファイルに異常が発生したと考えられる．

6 まとめ

本研究では，GitHub から Java プロジェクトと Android アプリケーションを抽出し，ビルドツールを用いたビルドの成功率や失敗原因などについて調査した．

3 つの RQ について調査を行い以下のような結論を得た．

- RQ1: 近年の Java プロジェクトおよび Android アプリケーションの多くでは自動的なビルドが機能していない．また，一部の環境においては，新しく，最近更新がなされたリポジトリのほうがビルドに失敗しやすい傾向が表れることが分かった．
- RQ2: Java プロジェクトと Android アプリケーションどちらも Java8 と Java11 の非互換性の影響を受けていることが分かった．また，Android アプリケーションがより強い影響を受けていることが分かった．
- RQ3: Android アプリケーションのビルドエラーを手動で分類した結果，file not found が原因の場合が最も多く，次いで依存関係が原因の場合が多いことが分かった．それらの対処方法について分類結果毎に考察した．

今後の課題として，Java や Kotlin 以外に対する調査が考えられる．Kotlin は Java と互換性のある言語である．Java と全く異なる言語のビルドがどのようになっているかは現在不明である．C 言語や Python などの現在広く用いられている言語や，web サイトなどで用いられる JavaScript など，異なる用途を持つ言語のビルドに対して調査を行うのは有意義だと考えられる．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 肥後 芳樹 教授にはご多忙の中，研究や発表における問題点の提示など，多くの御助言を賜りました．肥後 教授に厚く御礼申し上げます．

南山大学理工学部ソフトウェア工学科 井上 克郎 教授には研究の方針から研究の方針から本論文の執筆に至るまで，直接の御指導及び御助言を賜りました．井上 教授に厚く御礼申し上げます．

福知山公立大学情報学部情報学科　眞鍋 雄貴　講師には，研究の方針から本論文の執筆に至るまで，直接の御指導及び御助言を賜りました．眞鍋　講師に厚く御礼申し上げます．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授には研究や発表における問題点の提示など，多くの御助言を賜りました．松下 准教授に厚く御礼申し上げます．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 神田 哲也 助教には研究活動及び研究室での生活に多くの御助言を賜りました．神田 助教に厚く御礼申し上げます．

最後に，様々な御指導，ご助言を頂いたた大阪大学大学院情報科学研究科コンピュータサイエンス専攻肥後研究室の皆様に厚く御礼申し上げます．

参考文献

[1] mingc/android-build-box. https://hub.docker.com/r/mingc/android-build-box/.

[2] Carolin E. Brandt, Annibale Panichella, Andy Zaidman, and Moritz Beller. Logchunks: A data set for build log analysis. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR '20, p. 583–587, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Lorena P. de Figueiredo, Jonathan P. Gomes, Flavia de S. Santos, Guilherme M. Queiroz, Felipe T. Giuntini, and Juliano E. Sales. An automatic approach to detect problems in android builds through screenshot analysis. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22, p. 926–932, New York, NY, USA, 2022. Association for Computing Machinery.

[4] Martin Fowler. Continuous Integration. https://martinfowler.com/articles/continuousIntegration.html#BenefitsOfContinuousIntegration.

[5] Foyzul Hassan, Shaikh Mostafa, Edmund S.L. Lam, and Xiaoyin Wang. Automatic building of java projects in software repositories: A study on feasibility and challenges. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 38–47, 2017.

[6] Foyzul Hassan and Xiaoyin Wang. Hirebuild: An automatic approach to history-driven repair of build scripts. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 1078–1089, 2018.

[7] Noureddine Kerzazi, Foutse Khomh, and Bram Adams. Why do automated builds break? an empirical study. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 41–50, 2014.

[8] Yiling Lou, Zhenpeng Chen, Yanbin Cao, Dan Hao, and Lu Zhang. *Understanding Build Issue Resolution in Practice: Symptoms and Fix Patterns*, p. 617–628. Association for Computing Machinery, New York, NY, USA, 2020.

[9] Christian Macho, Shane McIntosh, and Martin Pinzger. Automatically repairing dependency-related build breakage. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 106–117, 2018.

[10] Andrew Neitsch, Kenny Wong, and Michael W. Godfrey. Build system issues in multi-language software. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pp. 140–149, 2012.

[11] Chaiyong Ragkhitwetsagul and Jens Krinke. Using compilation/decompilation to enhance clone detection. In *2017 IEEE 11th International Workshop on Software Clones (IWSC)*, pp. 1–7. IEEE, 2017.

[12] Matúš Sulír, Michaela Bačíková, Matej Madeja, Sergej Chodarev, and Ján Juhár. Large-scale dataset of local java software build results. *Data*, Vol. 5, No. 3, p. 86, 2020.

[13] Matúš Sulír and Jaroslav Porubän. A quantitative study of java software buildability. In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, pp. 17–25, 2016.

[14] Proksch Sebastian Zemp Timothy Gall Harald C. Vassallo, Carmine. Every build you break: developer-oriented assistance for build failure resolution. In *Empirical Software Engineering*, pp. 2218–2257, 2020.

[15] Joel Spolsky 著, 青木 靖訳. *Joel on Software*. オーム社, 2005.

Listing 8: 他の言語のコンパイルの例

```
066: FAILURE: Build failed with an exception.
067:
068: * What went wrong:
069: Execution failed for task ':app:mergeReleaseResources'.
070: > Multiple task action failures occurred:
071:     > A failure occurred while
      executing com.android.build.gradle.internal.res.Aapt2CompileRunnable
072:         > Android resource compilation failed
073:           ERROR:/tmp/build/Alfiyansya_GithubUserApp1/app/src/main/res/drawable/user3
      .png: AAPT: error: failed to read PNG
      signature: file does not start with PNG signature.
074:
075:           ERROR:/tmp/build/Alfiyansya_GithubUserApp1/app/src/main/res/drawable/user3
      .png: AAPT: error: file failed to compile.
076:
077:     > A failure occurred while executing com.android.build.gradle.internal.res
      .Aapt2CompileRunnable
078:         > Android resource compilation failed
079:           ERROR:/tmp/build/Alfiyansya_GithubUserApp1/app/src/main/res/drawable/user9
      .png: AAPT: error: failed to read PNG signature:
        file does not start with PNG signature.
080:
081:           ERROR:/tmp/build/Alfiyansya_GithubUserApp1/app/src/main/res/drawable/user9
      .png: AAPT: error: file failed to compile.
082:中略



106:
107:     > A failure occurred while executing
      com.android.build.gradle.internal.res.Aapt2CompileRunnable
108:         > Android resource compilation failed
109:           ERROR:/tmp/build/Alfiyansya_GithubUserApp1/app/src/
      main/res/drawable/user1.png: AAPT: error: failed to read PNG
      signature: file does not start with PNG signature.
110:
111:           ERROR:/tmp/build/Alfiyansya_GithubUserApp1/app/src/
      main/res/drawable/user1.png: AAPT: error: file failed to compile.
112:
113:
114: * Try:
115: Run with --info or --debug option to get more
      log output. Run with --scan to get full insights.
116:
```

Listing 9: ドキュメント生成の例

```
FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':spi-applike:javadoc'.
> Javadoc generation failed. Generated Javadoc options file
    (useful for troubleshooting):
    '/tmp/build/afirez_spi/spi-applike/build/tmp/javadoc/javadoc.options'

* Try:
Run with --info or --debug option to get more log output.
    Run with --scan to get full insights.
```

Listing 10: バージョン管理システムの例

```
FAILURE: Build failed with an exception.

* Where:
Build file '/tmp/build/GoodKitties_Fair/app/build.gradle' line: 4

* What went wrong:
An exception occurred applying plugin request
    [id: 'com.palantir.git-version', version: '0.11.0']
> Failed to apply plugin [id 'com.palantir.git-version']
    > Cannot find '.git' directory
```