修士学位論文

題目

Stack Overflow のコード片の進化パターンと そのOSSでの追従状況

指導教員
井上 克郎 教授

報告者
栗原 拓己

令和4年2月2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 3 年度 修士学位論文

Stack Overflow のコード片の進化パターンとその OSS での追従状況

栗原 拓己

内容梗概

ソフトウェア開発プラットフォームである GitHub では，プログラミングに特化した Q&A フォーラムである Stack Overflow からコード片の再利用が行われている．Stack Overflow のコード片は頻繁に変更が行われ，利用者はその修正状況を把握することが望ましい．しかし，Stack Overflow のコード片と再利用先の OSS のコード片はそれぞれ異なった変更が繰り返されており，2 つのプラットフォームで両者が活発に変更されているものはわずかであるという報告がなされている．そのため，Stack Overflow のコード片の進化を追従する必要性についてはより詳細な分析が必要である．本研究では，OSS が Stack Overflow のコード片の進化にどの程度追従しているのか調査する．また，OSS に利用されている Stack Overflow のコード片の変更履歴を分析し，進化パターンの分類を行い，進化に追従する必要性を調査する．対象は Java プロジェクトに限定し，データセット SOTorrnet を用いて再利用された Stack Overflow のコード片の変更履歴を手作業で調査した．結果として OSS に再利用されている Stack Overflow のコード片は 72.4%が最新バージョンのコード片，27.6%が旧バージョンのコード片であった．OSS の追従状況を調査したところ，Stack Overflow のコード片を再利用している OSS が Stack Overflow のコード片の進化に追従して共に進化している事例は発見されなかった．また，OSS に再利用されている Stack Overflow の旧バージョンが再利用後どのような変更がされているか調査したところ是正保守が 64 件，適応保守が 10 件，完全性保守が 55 件，機能追加が 24 件，非機能追加が 51 件であり，是正保守が最も多いことが分かった．OSS が SO のコード片の進化に追従している事例はなかったが，是正保守や完全性保守が多いため変更内容を把握して進化に追従する必要があると考えられる．

主な用語

OSS
ソフトウェア再利用
Stack Overflow

目 次

1 まえがき **3**

2 背景 **5**

2.1 Stack Overflow 5
2.2 SOTorrent 5
2.3 Stack Overflow の投稿の再利用に関する研究 6
2.4 Stack Overflow の投稿の品質に関する研究 7

3 調査方法 **10**

3.1 STEP1: 調査対象リストの作成 10
3.2 STEP2: OSS が再利用しているコード片のバージョンの分類 11
3.2.1 (A): URL を基にした再利用状況の調査 11
3.2.2 (B): コードクローン検出を用いた再利用状況の調査 12
3.3 STEP3: OSS の追従状況の調査 14
3.4 STEP4: 進化パターンの分類 14

4 調査結果 **16**

4.1 RQ1: OSS が SO のコード片の進化に追従して最新バージョンになっているものはどの程度あるのか 16
4.2 RQ2: 旧バージョンを再利用しているものは進化に追従する必要はあるのか . 17
4.2.1 是正保守 18
4.2.2 完全性保守 20
4.2.3 適応保守 21
4.2.4 機能追加 22
4.2.5 非機能追加 23
4.3 考察 24

5 妥当性への脅威 **26**

6 まとめ **27**

謝辞 **28**

参考文献 **29**

1　まえがき

ソフトウェア開発において，ソースコードの再利用は頻繁に行われている．特にソフトウェア開発プラットフォームである GitHub では，さまざまな Web サイトからコード片の再利用が行われている．その中でも，Stack Overflow のようなプログラミングに特化した Q&A フォーラムは盛んに利用されており，開発者はコーディングに関する質問をし，回答を求めることで効率よく開発を行っている．Stack Overflow では質問と回答が公開されているため，質問者以外の開発者も投稿に含まれるコード片を再利用することが多い．

Hata らの調査によると，GitHub プロジェクトのソースコード中にコメントとして記述される URL 中のドメインについて，Stack Overflow は 2 番目に多く参照されていることが明らかになっており，開発者が頻繁に Stack Overflow の投稿を参照していることが分かる [1]．しかし，Stack Overflow の投稿は修正などの目的で変更が加えられることが多く，投稿のコード片の再利用先で最新バージョンでない投稿が再利用され続けている可能性がある．Manes らの調査では，GitHub プロジェクトで参照されているコード片を含む Stack Overflow の投稿のうち 79%が，少なくとも 1 回は変更されていることが明らかになっている [2]．しかし，Manes らの異なる調査で，参照関係にある GitHub プロジェクトと Stack Overflow のコード片において，再利用後も活発に進化しているものは高々10%程度で，それ以外は独立に進化していることが明らかになっている [3]．この調査では両者のコード片の変更内容について詳細な分析が行われておらず，バグ修正などの再利用時に影響を及ぼすような Stack Overflow のコード片の変更を，コード片を再利用した GitHub プロジェクトの開発者が見逃している可能性が存在する．

そこで本調査では Java プロジェクトを対象として，OSS に再利用されている Stack Overflow のコード片の進化に関する調査を行い，進化のパターンについて分析を行う．また，これらのコード片の内容や OSS における追従状況に関しても調査し，Stack Overflow のコード片を再利用した際に，その後のコード片の進化に追従する必要性について考察する．本調査における “進化に追従する” とは Stack Overflow のコード片を OSS に再利用した場合，再利用後に再利用元の Stack Overflow のコード片が変更されたとき，その変更に合わせて OSS 側も同様に変更されることを指す．

本研究の RQ は以下の通りである．

RQ1: OSS が SO のコード片の進化に追従して最新バージョンになっているものはどの程度あるのか

RQ2: 旧バージョンを再利用しているものは進化に追従する必要はあるのか

以降，2 章では研究背景や関連研究について述べる．3 章では本研究の調査方法について述べ，4 章では，得られた結果と考察を述べる．5 章では本研究の調査結果に対する妥当性への脅威について述べる．最後に 6 章では本研究のまとめを述べる．

2 背景

2.1 Stack Overflow

今日，ソフトウェアの開発者がプログラミングの内容について何らかの問題に遭遇した場合，処理方法や API の使用法等をインターネット上で検索することは珍しくない．一般的に広く使われている検索方法として Google などの検索サービスの利用が挙げられるが，プログラミングに特化した質問サイトの利用も有用な選択肢の一つである．そのような質問サイトは，一般的な Web ページの検索よりも専門性や関連性の高い情報を発見しやすく，有識者からの回答を期待して自身の持つ疑問点を質問として投稿できるようになっている．代表的な質問サイトとして Stack Overflow が挙げられる．Stack Overflow は 2008 年に設立されたプログラミングに関する質問とそれに対する回答を投稿・検索できる Q&A フォーラムである [4]．2022 年 1 月現在，ユーザ数は約 1,700 万人，投稿は質問と回答を合わせて約 5,500 万件で，1 日に約 1,100 万人が Stack Overflow を閲覧しており，開発者に広く普及している [5]．開発者は，機能の実装方法や API の利用法などに関する疑問が生じた際に，その内容を文章やコード片を組み合わせて説明し質問として投稿する．質問に対する回答も同様に文章とコード片を組み合わせて投稿できる．

また，質問や回答に関するコメントや投稿内容の変更はすることができ，利用者は質問や回答の変更履歴を閲覧することができる．図 1 に回答の変更履歴の例を示す．回答例では 3 つのバージョンがあり，回答に対する 2 度目の編集では実線で囲まれたコード片において，赤でハイライトされたコードが削除され，緑でハイライトされた行が追加されていることが分かる．これらの編集は回答者だけでなくの他のユーザが変更可能であり，寄せられたコメントに関連した内容の変更も多い [6].

2.2 SOTorrent

SOTorrent [6] は，公式の SO データダンプと Google BigQuery GitHub データセットに基づくオーブンデータセットであり，投稿のテキストブロックやコードブロックごとの変更履歴が保存されている．コードブロックはソースコードのみで記述されたブロックで，テキストブロックは基本的にテキストで表現するがタグを使うことで文章中にコード片を組み込むことができる．図 2 にテキストブロックとコードブロックに分けた Stack Overflow の回答例を示す．図 2 で破線で囲まれている番号 1,3 のブロックがテキストブロック，実線で囲まれている番号 2 はコードブロックである．番号 4 はこの回答に寄せられたコメントである．SOTorrent を用いることでそれぞれのブロックごとのバージョン履歴にアクセスでき，詳細

[1] https://stackoverflow.com/posts/3537085/revisions

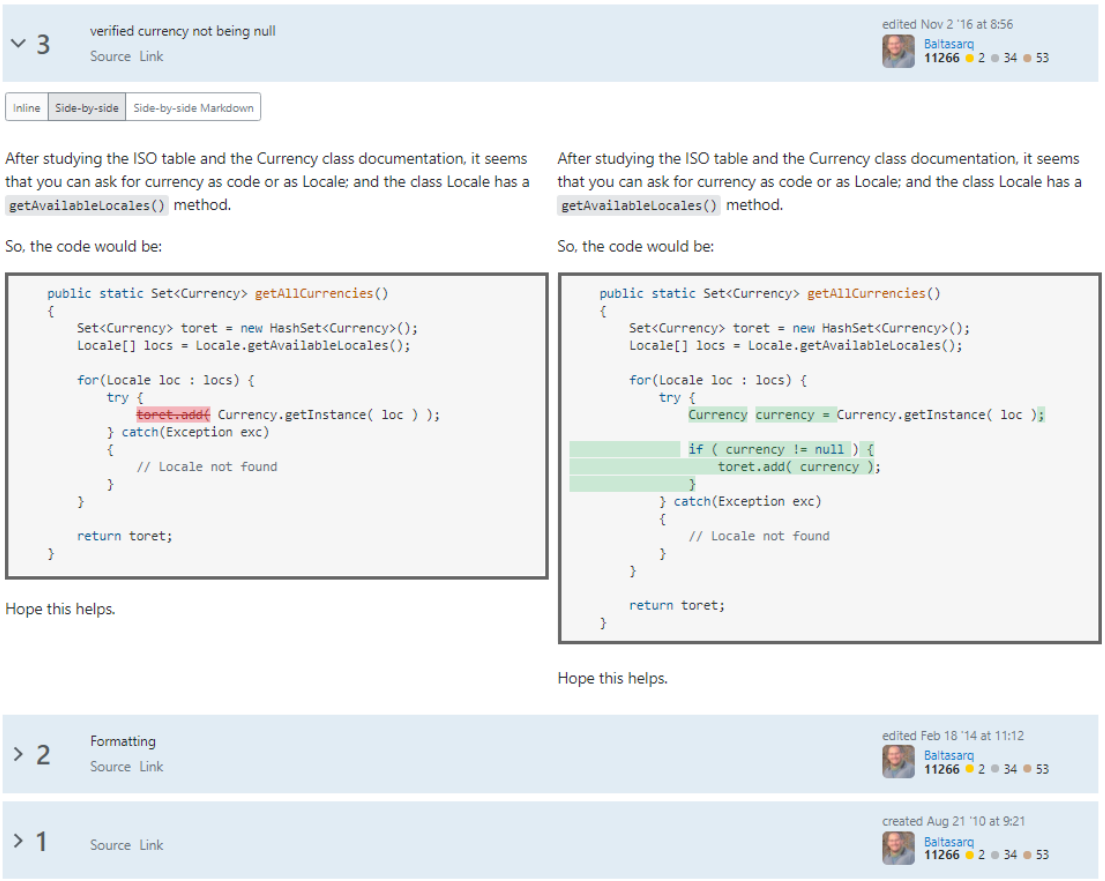

図 1: Stack Overflow の投稿の編集履歴（ID:3537085）[1]

な分析が可能である．また，SOTorrent には Stack Overflow の投稿を参照した GitHub プロジェクトの URL などの情報も格納されている．この情報を用いることで，GitHub プロジェクトにおける Stack Overflow のコード片の利用状況を調査することができる．

2.3 Stack Overflow の投稿の再利用に関する研究

ソフトウェアの再利用とはソフトウェア開発に必要とされるプログラムやデータをパターン化し，部分的に切り抜くことで繰り返し利用することである．ソフトウェア開発における機能の実装の際に，すでに存在する他のソフトウェアや Q&A サイトなどからソースコードを再利用することは一般的に行われており，Stack Overflow の投稿のコード片も開発者に再利用されることが多い．

Wu らは GitHub プロジェクトにおける Stack Overflow の投稿の再利用に関する調査を行

[2] https://stackoverflow.com/questions/3536968/get-all-possible-available-currencies/3537085#3537085

図 2: ブロックごとに分けられた Stack Overflow の回答例（ID:3537085）[2]

い，49%の投稿が編集を伴って再利用されている事を明らかにしている [7]．Stack Overflow のコード片が再利用される目的としてはプロジェクトの機能追加や機能向上などがあげられる [8]．また OSS 開発者は時間やスキルなどのリソースが限りがあるが，機能を迅速に統合でき，開発コストが軽減できるなどの理由からコード片の再利用を行っている [9]．

2.4 Stack Overflow の投稿の品質に関する研究

Stack Overflow の投稿のコード片の利用における問題も指摘されている．複数の研究で，Stack Overflow などの Q&A サイトのコード片を利用することにより，バグの混入 [10]，ライセンス違反 [11] やセキュリティの脆弱性 [12] の問題を引き起こす恐れがあることが示されている．また，Zhang らは Stack Overflow における情報の陳腐化について調査を行って

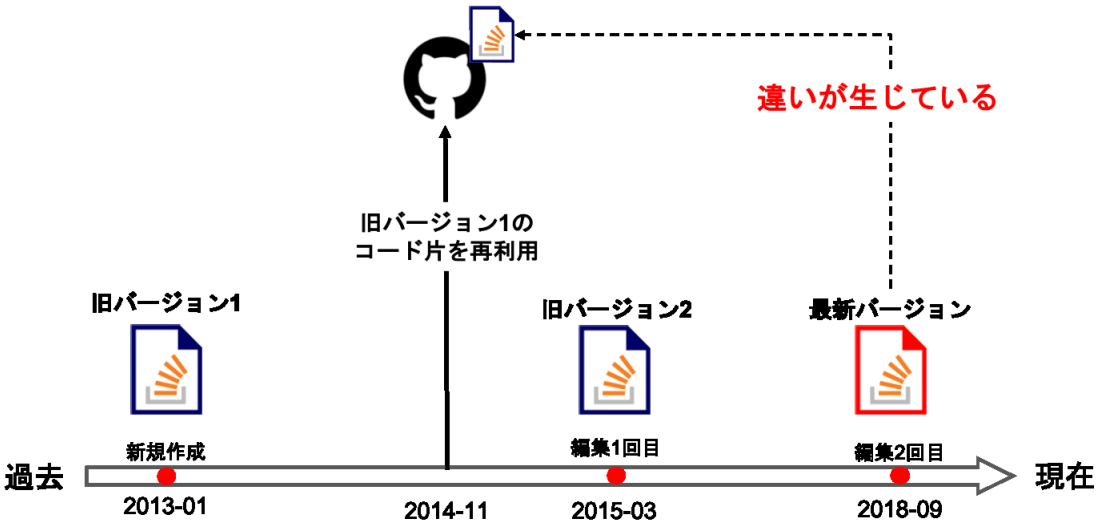

図 3: Stack Overflow の旧バージョンのコード片を再利用している例

いる [13]．Stack Overflow において陳腐化した回答の半数以上が最初に投稿された時点で古い情報であり，そのような回答のうちその後変更されたものは 20.5%のみであった．また，陳腐化に対する指摘まで平均して 118 日要しており，陳腐化した投稿の修正には時間を要することが分かっている．

GitHub プロジェクトで Stack Overflow のコード片を再利用する際，再利用時点からコード片が変更されると OSS は進化に追従せずバージョンの違いが生じる．具体的には Stack Overflow のコード片を GitHub プロジェクトで利用した後，参照元の Stack Overflow のコード片が変更される場合がある．そのような事例の概要を図 3 に示す．この図では Stack Overflow の新規投稿として旧バージョン 1 のコード片が作成されている．その後 GitHub プロジェクトは Stack Overflow の旧バージョン 1 のコード片を再利用している．しかし，その後 Stack Overflow は旧バージョン 2 と最新バージョンと 2 回コード片が変更がされている．このような場合，GitHub プロジェクトで再利用した Stack Overflow のコード片と Stack Overflow の最も新しいバージョンのコード片との間には違いが生じてしまう．

このような事例に対する調査として，Manes らの研究では Stack Overflow の投稿とその投稿から GitHub プロジェクトで再利用されるコード片についてそれぞれの進化に着目して調査を行っている．結果として，Stack Overflow では 76%のコード片が進化しているのに対し，GitHub プロジェクトへ再利用されたコード片は 22%しか進化していないことがわかった．また 2 つのプラットフォームでともに進化しているものは高々10%程度であることを明らかにしている [3]．

以上の既存研究から，Stack Overflow のコード片は GitHub プロジェクトにおいて頻繁に

再利用されているが，バグや脆弱性，さらには陳腐化した情報も多い上，これらの修正には時間を要することが分かっている．これらの問題があるコード片を OSS が再利用しており，Stack Overflow の投稿においてこれらの問題が修正されていた場合は，同様の修正を GitHub プロジェクトにおいても反映すべき可能性があるが，現状そのような事例は少ないことが分かる．ソフトウェア開発において品質が重視される中，他プラットフォームなどの外部リソースからのコード片の再利用は，再利用時点では問題の解決策として得策である場合もあるが，ソフトウェアのその後の開発効率，機能性，保守性に影響を与える可能性があるため，開発者は十分に注意する必要がある．

3　調査方法

本研究では，Stack Overflow のコード片を OSS に利用する際に，再利用後 OSS は Stack Overflow のコード片の進化に追従しているか調査する．また．再利用されたコード片は Stack Overflow のコード片の進化に追従する必要性があるか調査する．本調査では Stack Overflow，OSS 共に Java で書かれたコード片，ソースコードを対象とする．

それぞれ以下の 2 つの Research Question を設定した．

- RQ1: OSS が Stack Overflow のコード片の進化に追従して最新バージョンになっているものはどの程度あるのか
- RQ2: 旧バージョンを再利用しているものは進化に追従する必要はあるのか

RQ1 は OSS に Stack Overflow の最新バージョンが再利用されている場合，OSS が Stack Overflow の旧バージョンを再利用しており Stack Overflow の変更後，Stack Overflow の進化に追従して最新バージョンが再利用されている可能性がある．そのような事例がどの程度あるのか調査する．RQ2 は OSS が追従していない Stack Overflow の旧バージョンのコード片に対して進化に追従する必要性を調査する．そのために OSS にコード片が再利用された後 Stack Overflow のコード片はどのように変更されているか 5 つの進化パターンに分類し，進化に追従する必要性について考察する．

これらの Research Question に回答するために以下の STEP1 から STEP4 の手順に従って，本調査を行う．

STEP1: 調査対象リストの作成

STEP2: OSS が再利用しているコード片のバージョンの分類

(A) URL を基にした再利用状況の調査

(B) コードクローン検出を用いた再利用状況の調査

STEP3: OSS の追従状況の調査

STEP4: 進化パターンに分類

各ステップの詳細について説明する．

3.1　STEP1: 調査対象リストの作成

STEP1 では SOTorrent からコードブロックが 1 回以上変更された Stack Overflow の Java に関する回答を抽出し調査対象リストを作成する．本調査ではデータセットとして Baltes [6]

らの研究で作成された SOTorrent(2018-09-23) を使用し，データベース構築には Diamantopoulos [14] らの研究で使用されたスクリプトを使用する．このスクリプトは SOTorrent の Java の投稿 (Posts テーブルの Tags カラムに “java” タグが含まれるもの) のみを抽出する．Diamantopoulos らが研究で使用したスクリプトでは Tags カラムに “java” タグが含まれているもののみを抽出しているが，その場合，再利用先の OSS のファイルとして Java 以外の JVM 系言語である Scala や Kotlin のファイルが含まれてしまう．これらの JVM 系の言語では，Java に関する回答の知識を利用して開発は可能であるものの，コード片の利用のためはそれぞれの言語に応じて修正が必須なため，今回の調査目的から考慮すると調査対象から除外する必要がある．そのため抽出した OSS のファイルから拡張子が “java” 以外のものは除外するようにスクリプトを拡張した．

3.2 STEP2: OSS が再利用しているコード片のバージョンの分類

STEP2 では Stack Overflow のコード片を再利用している OSS に対して，Stack Overflow のコード片の最新バージョンを再利用しているか旧バージョンを再利用しているか 2 つに分類する．STEP2(A) では URL を基に再利用状況を手作業で調査する．Baltes [15] らの研究によると Stack Overflow から GitHub にコード片の再利用されたもののうち参照元の質問や回答を記述しているものは実際に再利用されたもののうち 25%程度と推測される．また，アンケートを行ったソフトウェア開発者のうち，約半数が参照元の記述なしに Stack Overflow からコード片を再利用していることを認め，約 3 分の 2 の開発者がコード片のライセンスとその内容について認識していないということが分かっている．このことから OSS に Stack Overflow のコード片が，参照元つまり URL を記述せずに再利用されている可能性がある．そのため STEP2(B) では調査対象リストから最新バージョンのコード片と旧バージョンのコード片をリスト化しそれぞれと OSS の間でコードクローン検出を行い，類似したコード片を見つけることで参照元が記述されてないものに対しても調査を行った．

3.2.1 (A): URL を基にした再利用状況の調査

STEP2(A) では STEP1 で作成した調査対象リストから Stack Overflow の回答の URL とその回答を参照している GitHub プロジェクトのファイルの URL の組を抽出した．同じプロジェクトからフォークあるいは複製されたプロジェクトの影響を除外するため，Stack Overflow から複数の GitHub プロジェクトに利用されている場合，Stack Overflow の投稿 1 件につき，1 つの GitHub プロジェクトを調査した．その URL を基に，GitHub プロジェクトのコミット履歴と Stack Overflow のコード片の変更内容を見比べ，最新バージョンを再利用しているか旧バージョンを再利用しているか手作業で 2 つに分類する．STEP2(A) の調査事例を図 4 に示す．

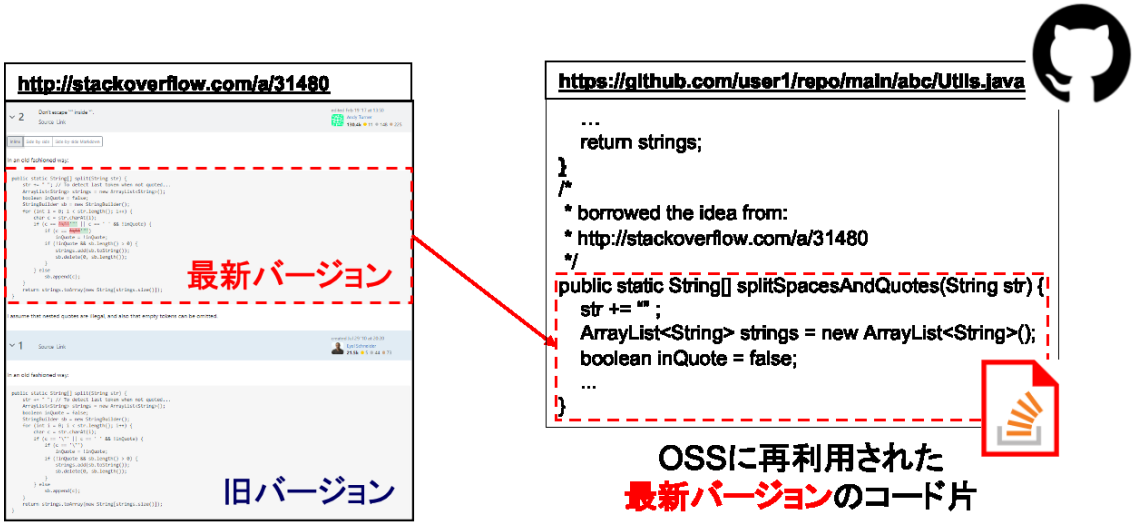

図 4: STEP2(A) の調査事例

抽出した URL は Stack Overflow の URL は “http://stackoverflow.com/a/31480”，GitHub プロジェクトのファイルの URL は “https://github.com/user1/repo/main/abc/Utils.java” になっている．GitHub プロジェクトのファイルには Stack Overflow の URL が記述されており，どのコード片から Stack Overflow を参照したかを確認することができる．これを基に Stack Overflow のコード片の各バージョンを確認してどのバージョンを再利用しているか調査する．例の場合だと Stack Overflow の最新バージョンのコード片を再利用している．

3.2.2 (B): コードクローン検出を用いた再利用状況の調査

STEP2(B) では最新バージョンと旧バージョンの利用状況を調査するために，STEP1 で作成した調査対象リストから Stack Overflow の各投稿の最新バージョンのコード片と旧バージョンのコード片のリストをそれぞれ作成する．次に最新バージョンのリストと旧バーションのリストそれぞれと 22 個の Android アプリケーションの間でコードクローン検出を行う．今回対象とする Android アプリケーションのソースコードは F-Droid [16] から取得した．F-Droid は Android アプリケーションストアであり，公開されているアプリケーションは Android 専用のソフトウェアパッケージのファイルフォーマットの APK ファイルだけでなく，各アプリケーションのソースコードへのリンクが公開されている．Android アプリケーションの詳細を表 1 に示す．

コードクローン検出器には CCFinderX [17] を使用する．コードクローンには 4 つのタイプが存在し，タイプ 1 は空白・改行・コメントなどのコーディングスタイルを除いて一致する

表 1: STEP2(B) の対象の Android アプリケーション

アプリケーション名	コミット数	コントリビュータ数	LOC
OsmAnd	72,760	794	426,256
FrostWire	6,709	21	147,997
Open Explorer	1,669	10	130,565
WordPress	60,688	160	97,620
AnkiDroid	13,130	146	89,060
Xabber Classic	4,264	23	69,643
Smart Receipts Pro-	2,495	9	50,945
Barcode Scanner	3,588	98	46,614
F-Droid	7,132	95	44,699
APG Encrypt	4,367	74	41,993
Andlytics Track	1,524	40	16,694
ForPDA	623	4	15,222
Open Training	503	6	10,289
BeTrains NMBS/SNCB	344	7	10,213
YASRA	21	2	9,128
Secrecy Secure file storage	155	1	6,549
Tram Hunter	288	7	5,516
OpenDocument Reader	4,971	8	4,840
OpenLaw	333	1	4,037
OCR Test	115	1	3,929
blippex	18	3	2,050
AnagramSolver	50	2	745

もの，タイプ 2 はタイプ 1 の違いに加えて，リテラルや識別子などの違いを除いて一致するもの，タイプ 3 はタイプ 2 にの違いに加えて，文の挿入・削除などの違いを除いて一致するもの，タイプ 4 は構文的には一致しないが同様の処理を行うものとなっている．CCFinderX はタイプ 1 とタイプ 2 のコードクローンを検出するために開発された字句単位のコードクローン検出器である．今回は進化に関係があるか否かを明確にするためにタイプ 1，タイプ 2 の検出を目的としているため CCFinderX を使用した．CCFinderX のコードクローン検出するトークン数は 60 に設定した．調査事例を図 5 に示す．まず 1 回以上変更されている Java 言語で書かれたコードブロックを各バージョンごとにコード片として切り出す．そしてそれぞれと CCFinderX を用いて Android アプリケーションの間でコードクローン検出を行う，結果としてコードクローンが見つかった場合は，それは OSS に再利用されているコード片であることがわかる．

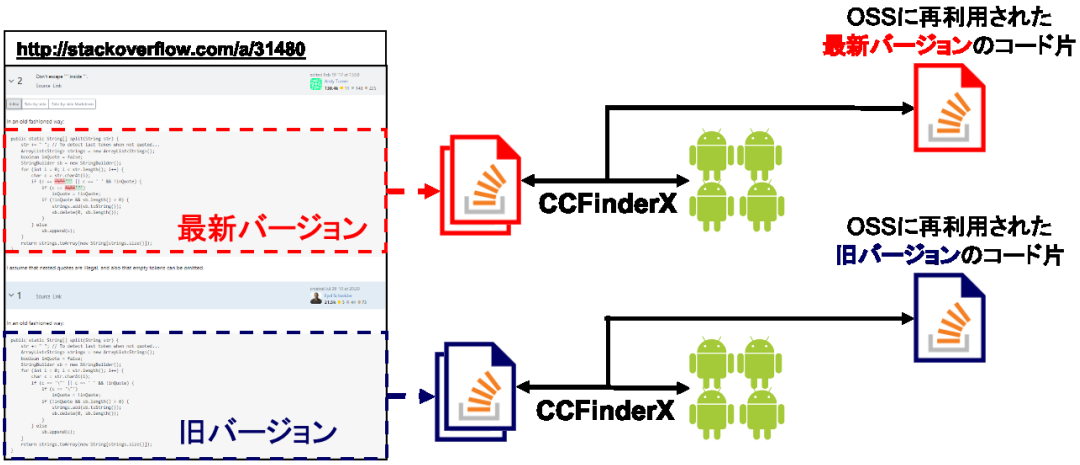

図 5: STEP2(B) の調査事例

3.3 STEP3: OSS の追従状況の調査

STEP3 では STEP2 で見つかった OSS が Stack Overflow の最新バージョンのコード片を再利用しているものに対して，最新バージョンを再利用している OSS の変更履歴と再利用されているコード片の変更履歴を遡り，OSS が Stack Overflow のコード片の進化に追従している事例がどの程度あるのか調査する．追従している例として OSS は Stack Overflow の旧バージョンのコード片を採用してその後 Stack Oveflow の進化に追従して最新バージョンのコード片に変更したことになる．そのため Stack Overflow の新規投稿から最新バージョンの間にコミット履歴がある OSS のファイルを調べる必要がある．それらのファイルに対して変更履歴を遡り，進化に追従しているか調べる．

3.4 STEP4: 進化パターンの分類

STEP4 では STEP2 で旧バージョンを利用していると分類されたものに対して，OSS が Stack Oveflow のコード片を再利用した後に Stack Overflow のコード片が変更された後，どのように進化しているか 5 つの進化パターンに分類し進化に追従する必要性を調査する．本調査では Hindle ら [18] の研究で用いられた進化パターンを拡張して利用した．進化パターンは

- 是正保守
- 適応保守
- 完全性保守

- 機能追加
- 非機能追加

の 5 分類である．

是正保守はプログラムが異常終了してプロセスがキャンセルされたり，設計段階で指定された性能基準を満たしていないものに対して行う変更である．適応保守はデータやプロセスの環境の変化に対応した変更であり，例として，データベースの再構築や，OS のバージョンが変更される場合などがあげられる．完全性保守は処理の非効率性を解消し，機能を向上させ保守性を改善するために行うものである．この変更が行われる理由としては計算アルゴリズムの改善や実行時間の短縮などがあげられる．機能追加は既存のソースコードには無い仕様を追加する変更である．非機能追加はコードの整形など Hindle らの定義に加えて，Stack Overflow 特有の変更であるクラス宣言の追加など処理自体の変更とは関係ないものである．

OSS が Stack Overflow の旧バージョンのコード片を利用している場合，採用されたバージョンから最新のバージョンを比較し，編集者のコメントを参考に変更を筆者と協力者の 2 名で 5 つの進化パターンに分類する．複数の分類に当てはまる変更が行われていた場合は，1 つの変更に対して複数種類のラベル付けを行った．本調査ではコードブロックが 1 回以上変更されている回答が対象となっており，コードコメントのみの追加や変更がされている場合は調査対象外として除外した．

表 2: STEP2(A) における再利用されているコード片のバージョンの調査結果

利用状況	件数
最新バージョン	186
旧バージョン	46
リンクが無効	31

表 3: STEP2(B) における再利用されているコード片のバージョンの調査結果

利用状況	件数
最新バージョン	278
旧バージョン	131

4 調査結果

4.1 RQ1: OSS が SO のコード片の進化に追従して最新バージョンになっているものはどの程度あるのか

3.2.1 節で説明した手順に従って得られたコードブロックが 1 回以上変更されている Stack Overflow のコード片とそのコード片を再利用している OSS のファイルの組は 263 件であった．その結果を表 2 に示す．これらの組に対して OSS で再利用されている Stack Overflow コード片が最新バージョンであるか旧バージョンであるか 2 つに分類をした．263 件中 31 件について OSS がプライベートリポジトリに設定された，または，GitHub プロジェクトのリポジトリが削除されていたため，Stack Overflow のどのバージョンのコード片を再利用しているか調査できなかった．調査可能な 232 件のうち 186 件が最新バージョン，46 件が旧バージョンを再利用していた．

3.2.2 節で説明した手順に従って Android アプリケーションと Stack Overflow の最新バージョン，旧バージョンそれぞれのコード片のリストの間でコードクローン検出を行い，類似したコード片を見つけることで，参照元が記述されてないものに対しても調査を行った．その結果を表 3 に示す．結果として Android アプリケーションと最新バージョンとの間で検出されたコードクローンは 278 件，旧バージョンとの間で検出されたコードクローンは 131 件であった．

STEP2(A) と STEP2(B) の結果を合計すると OSS で Stack Overflow の最新バージョンを再利用しているものが 464 件，旧バージョンを再利用しているものが 177 件であった．Stack Overflow の最新バージョンのコード片が再利用されているもしくは，最新バージョンのコード片が検出されている OSS に対して参照元のコード片の進化に追従しているものを調査し

表 4: URL を基にした進化パターン分類結果

進化パターン	件数
是正保守	11
適応保守	6
完全性保守	15
機能追加	2
非機能追加	12

表 5: コードクローン検出を用いた進化パターン分類結果

進化パターン	件数 (件)
是正保守	53
適応保守	4
完全性保守	40
機能追加	22
非機能追加	39

たところそのような事例は見られなかった.

RQ1

OSS に再利用されている Stack Overflow のコード片は 72.4%が最新バージョンであり，27.6%が旧バージョンであった．また，OSS が Stack Overflow のコード片の進化に追従している例は見られなかった.

4.2 RQ2: 旧バージョンを再利用しているものは進化に追従する必要はあるのか

OSS が再利用した旧バージョンの Stack Overflow のコード片が再利用後に変更された場合，コード片の進化パターンを是正保守，適応保守，完全性保守，機能追加，非機能追加の 5 つに分類した.

調査対象は STEP2(A) で見つかった Stack Overflow の旧バージョンのコード片を再利用していた 46 件である．表 4 に進化パターンの分類結果を示す．STEP2(B) で見つかった分類対象は Android アプリケーションと Stack Overflow の旧バージョンのコード片のリストで行ったコードクローン検出によって得られた 131 件である．表 5 に進化パターンの分類結果を示す.

是正保守は URL を基にした調査では 11 件，コードクローン検出を用いた調査では 53 件あり，変更前のコード片だと回答に対する機能は実装できていないため編集されたものだっ

た．適応保守は URL を基にした調査では 6 件，コードクローン検出を用いた調査では 4 件あり，言語のバージョンやライブラリへの依存を解消するものだった．完全性保守は URL を基にした調査では 15 件，コードクローン検出を用いた調査では 40 件あり，処理を高速化するためのメソッドの変更や，リファクタリングなどの目的で変更されていた．機能追加は URL を基にした調査では 2 件，コードクローン検出を用いた調査では 22 件あり，質問者が投稿した内容に関する実装方法にテストコードを追加する変更などがあった．非機能追加は URL を基にした調査では 12 件，コードクローン検出を用いた調査では 39 件であった．実装には影響しないが質問者に対する説明補足としてコンパイル可能なコード片に変更されていた．またコード片内のパスの変更やコードの整形による変更などがあった．

それぞれの進化パターンについて例を示し，どのような進化パターンであれば進化に追従する必要性があるか考察した．コード片の行頭に「+」がついているものはその変更により追加された行であり,「-」がついているものはその変更により削除された行である．

4.2.1 是正保守

投稿 ID：7322581

Java を用いた大きなファイルからテキストの最終行を高速で読み取る方法についての質問に対して，回答として以下のようなコードブロックが寄せられた．

```
public String tail2( File file, int lines) {
        ...
            if( readByte == 0xA ) {
-                line = line + 1;
-                if (line == lines) {
-                    if (filePointer == fileLength) {
-                        continue;
-                    }
-                    break;
+                if (filePointer < fileLength) {
+                    line = line + 1;
+                }
            }
            } else if( readByte == 0xD ) {
-                line = line + 1;
-                if (line == lines) {
-                    if (filePointer == fileLength - 1) {
-                        continue;
-                    }
-                    break;
-                }
```

```
+                 if (filePointer < fileLength-1) {
+                     line = line + 1;
+                 }
+             }
+             if (line >= lines) {
+                 break;
             }
             sb.append( ( char ) readByte );
         ...
}
```

変更前のコード片では改行コード \r \n のあるファイルでは正しく機能しない．改行コード \r \n を使用すると，行数をカウントする条件 line == lines をスキップして全体を読み取ることになる．よって改行コード \r \n の場合，1 行ごとに 2 行カウントされてしまうので，変更後のコード片では，最終行の N 行を取得するために，n*2 行を指定するように変更されている．これはコード片を利用する際に発生するバグの修正を行なっており，是正保守に分類される．

投稿 ID：35989142

Java.nio パッケージを使用して最も簡単に再帰的にディレクトリを削除する方法についての質問に対して，回答として以下のようなコードブロックが寄せられた．

```
Path rootPath = Paths.get("/data/to-delete");
// before you copy and paste the snippet
        ...
// the snippet below
+ try (Stream<Path> walk = Files.walk(rootPath)) {
+     walk.sorted(Comparator.reverseOrder())
    Files.walk(rootPath)
        .sorted(Comparator.reverseOrder())
        .map(Path::toFile)
        .peek(System.out::println)
        .forEach(File::delete);
+ }
```

ここではファイルシステムのリソースをタイムリーに廃棄する必要があるため，例外処理として try-with-resources 文が追加されている．このコードの変更は例外において正常に動作するようにした変更であるため，是正保守に分類される．

4.2.2　完全性保守

投稿 ID：3775723

Java を使用してファイルを削除する方法の質問に対して，回答として以下のようなコードブロックが寄せられた．

```
public static boolean deleteDir(File dir) {
 if (dir.isDirectory())
 {
  String[] children = dir.list();
  for (int i=0; i<children.length; i++)
- {
-  boolean success = deleteDir(new File(dir, children[i]));
-  if (!success)
-  {
-   return false;
-  }
- }
+ return deleteDir(new File(dir, children[i]));
 }
// The directory is now empty or this is a file so delete it
 return dir.delete();
}
```

変更前は boolean 型のメソッドの返り値を変数に代入して条件分岐に使っていたが，変更後は変数への代入を行わずに処理を行っている．これはコード片のリファクタリングを行っており，変更によって本来の処理の内容が変化することはない．よって完全性保守に分類される．

投稿 ID：2222268

サーブレットに対して送られてきたリクエストに関して，正確な URL を取得するには HttpServletRequest インターフェースのオブジェクトを使用する．その使用方法の質問に対して，回答として以下のようなコードブロックが寄せられた．

```
public static String getFullURL(HttpServletRequest request){
-   StringBuffer requestURL = request.getRequestURL();
+   StringBuilder requestURL = new StringBuilder(request
+   .getRequestURL().toString());
    String queryString = request.getQueryString();

    if (queryString == null) {
        return requestURL.toString();
```

```
    } else {
        return requestURL.append('?').append(queryString)
        .toString();
    }
}
```

ここでは，StringBuffer を StringBuilder に変更している．StringBuffer はスレッドセーフであり，StringBuilder はスレッドセーフではない．コード片を利用しているプロジェクトでは，この変更によりパフォーマンスが向上するが，本来の処理内容が変化することはなく，完全性保守に分類される．

4.2.3 適応保守

投稿 ID：22492582

この質問は AES-256 ビット暗号化を使用したアプリ開発に関する質問である．質問者は AES-256 ビット暗号化を使用したアプリをユーザがインストールする際，JCE(Java Cryptography Extension) の無制限暗号強度管轄ポリシーファイルをインストールしてそのファイルと書き換える必要があるが，その作業はユーザーにとって手間であるため，他の方法がないか質問している．回答として以下のコードブロックが寄せられた．

```
private static void removeCryptographyRestrictions() {
    if (!isRestrictedCryptography()) {
        logger.fine("Cryptography restrictions removal not needed");
        return;
    }
    try {
        ...
    } catch (final Exception e) {
        logger.log(Level.WARNING, "Failed to remove cryptography restrictions",
        ↪  e);
    }
}

private static boolean isRestrictedCryptography() {
-     // This simply matches the Oracle JRE, but not OpenJDK.
-     return "Java(TM) SE Runtime
↪  Environment".equals(System.getProperty("java.runtime.name"));
+     // This matches Oracle Java 7 and 8, but not Java 9 or OpenJDK.
+     final String name = System.getProperty("java.runtime.name");
+     final String ver = System.getProperty("java.version");
+     return name != null && name.equals("Java(TM) SE Runtime Environment")
```

```
+             && ver != null && (ver.startsWith("1.7") ||
↪  ver.startsWith("1.8"));
}
```

こちらでは，アプリをインストールして使用する際，JCE の無制限暗号強度管轄ポリシーファイルをダウンロードする以外の方法を提案している．その後回答では Java 9 でも動作が可能になるように変更がされている．これは処理環境に関する変更であり，適応保守に分類される．

4.2.4 機能追加

投稿 ID：9520650

この質問は入力された値を取得することができる EditText クラスを実装することに関する質問である．質問者は 100 以上の入力があった場合，入力された値を強制的に 100 にするという実装について質問し，回答者はその方法をコードブロックと共に回答として投稿している．その後回答者は，その実装のテストコードとして以下のようなコードブロックを追加した．

```
+ public class CreateNewForm extends Activity {
+     EditText  CaseAge;
+     @Override
+     protected void onCreate(Bundle savedInstanceState) {
+         super.onCreate(savedInstanceState);
+         LinearLayout
+                 ll = new LinearLayout(this);
+         CaseAge = new EditText(this) ;
+         CaseAge.setWidth(69);
+         ll.addView(CaseAge);
+         setContentView(ll);
+         CaseAge.setFilters(new InputFilter[]{new InputFilterMinMax(1, 100)});
+     }
+     class InputFilterMinMax implements InputFilter {
+         private int min, max;
+         public InputFilterMinMax(int min, int max) {
+             this.min = min;
+             this.max = max;
+         }
+         public CharSequence filter(CharSequence source, int start, int end,
↪  Spanned dest, int dstart, int dend) {
+             try {
```

```
+                 int input = Integer.parseInt(dest.toString() +
↪  source.toString());
+                 if (isInRange(min, max, input))
+                     return null;
+             } catch (NumberFormatException nfe) {
+             }
+             return "";
+         }
+         private boolean isInRange(int a, int b, int c) {
+             return b > a ? c >= a && c <= b : c >= b && c <= a;
+         }
+     }
+ }
```

変更点としては，質問者が実装したい機能を回答としてコード片を残した後，過去になかったテストコードを新たに追加している．これは新たな機能を追加しているため，機能追加に分類される．

4.2.5 非機能追加

投稿 ID：10174938

指定された文字列が Java で有効な JSON であるかどうかを確認する方法の質問に対して，回答として以下のようなコードブロックが寄せられた．

```
+ import org.json.*;
public boolean isJSONValid(String test) {
    try {
        new JSONObject(test);
    } catch (JSONException ex) {
        // edited, to include @Arthur's comment
        // e.g. in case JSONArray is valid as well...
        try {
            new JSONArray(test);
        } catch (JSONException ex1) {
            return false;
        }
    }
    return true;
}
```

変更点としては，import 文が追加されている．ここでは GitHub，maven，Android で利用可能な org.json JSONAPI 実装を使用するように説明補足されている．これは開発者がよ

りこのコード片を利用しやすいように変更されているが，コード片を利用している OSS には直接影響しないものであり，非機能追加に分類される．

投稿 ID：2248203

Java の ThreadPoolExecutor クラスを使用して，高負荷のタスクを固定数のスレッドで実行する際の例外処理の方法に関する質問に対して，回答として以下のようなコードブロックが寄せられた．

```
+ public final class ExtendedExecutor
+ extends ThreadPoolExecutor {
+     // ...
        ...
    protected void afterExecute(Runnable r, Throwable t) {
          super.afterExecute(r, t);
        ...
    }
+ }
```

ここでは，メソッドがどのクラスを継承することで利用可能かを開発者に分かりやすく示すために，クラス宣言が追加されている．このコード片を参照している GitHub プロジェクトにおいても該当クラスの継承を利用しており，GitHub プロジェクトに直接影響する変更ではないため非機能追加に分類される．

OSS で Stack Overflow の旧バージョンを再利用しているものに対して再利用後 Stack Overflow のコード片の変更を 5 つのパターンに筆者ら 2 人で分類した．URL を基にした調査とコードクローン検出を用いた調査の結果を合計すると是正保守が 64 件，適応保守が 10 件，完全性保守が 55 件，機能追加が 24 件，非機能追加が 51 件であった．最も多い変更は是正保守であったが，完全性保守や非機能追加も変更目的としては多く見られた．

RQ2

OSS に再利用されている Stack Overflow の旧バージョンを進化パターンに分類した結果，是正保守が 64 件，適応保守が 10 件，完全性保守が 55 件，機能追加が 24 件，非機能追加が 51 件であり是正保守が最も多かった．

4.3 考察

RQ1 では OSS が Stack Overflow のコード片を再利用している Stack Overflow のコード片のうち 27.6%がすでに変更がされている旧バージョンであることが分かった．また OSS が Stack Overflow の最新バージョンのコード片の進化に追従している例は発見されなかった．RQ2 では OSS に再利用されている Stack Overflow の旧バージョンのコード片が再利用

後にどのような変更されているか調査したところ，是正保守が 64 件，適応保守が 10 件，完全性保守が 55 件，機能追加が 24 件，非機能追加が 51 件であり，是正保守が最も多いことが分かった．是正保守は Stack Overflow の質問に対する回答として機能が十分でないときの変更や修正バグを含んでいる事例があり，OSS に再利用する開発者は OSS が再利用している Stack Overflow の進化を追い，その変更内容を把握して追従すべきである．完全性保守は OSS が Stack Overflow のコード片の進化に追従しなくてもバグが発生するということはなく，是正保守に比べて即座に進化に追従する必要性は薄いが，パフォーマンスの向上やリファクタリングの目的で変更されているものがあるため，Stack Overflow の変更には注意を払う必要がある．非機能追加は説明補足による変更などのため OSS 開発者は変更履歴を追う必要性は低い．是正保守の目的で編集されている Stack Overflow のコード片にバグを追加する変更になっているものが発見された．このように Stack Overflow のコード片の変更が必ずしも改善に向かうわけではない．そのため，機械的に追従すべきと断定することは難しいが，現状，追従が行われていないことを考えると OSS 開発者に変更されていることを通知することは有益であると考える．

5　妥当性への脅威

内部妥当性への脅威については 5 つの進化パターンに分類を行う際，Stack Overflow の変更時のコメントや変更内容を確認して分類を行ったが本来編集者の目的とは異なる進化パターンに分類している可能性がある．また，投稿の編集者がコメントを簡略化している可能性があるため，本来の意図を汲み取ることは難しい．

外部妥当性への脅威については調査対象の OSS が Java 言語に限られているため，他の言語においても同様の結果が得られるかは，Stack Overflow のコード片の変更頻度や内容，言語ごと固有の特徴により結果が変わる恐れがある．

6　まとめ

本調査では，OSS が再利用した Stack Overflow のコード片の進化に追従する必要性を調査するため，実際の再利用の事例に基づいて詳細な分析を行った．

具体的にはデータセットとして SOTorrent を用いて OSS に再利用されている Stack Overflow のコード片の変更履歴を分析し，進化パターンの分類を行い，進化に追従する必要性について調べた．OSS が再利用しているコード片のバージョンを調べるため URL に基づいた調査とコードクローン検出を用いた調査を行った．URL に基づいた調査では SOTorrent から Stack Overflow のコード片とそのコード片を再利用している GitHub プロジェクトのソースコードの URL の組を抽出し，それぞれの利用状況と再利用後の進化パターンの分類を行った．コードクローン検出を用いた調査では SOTorrent から Stack Overflow の最新バージョン，旧バージョンのコード片のリストを作成し，22 個の Android アプリケーションとの間でそれぞれ CCFinderX を用いてコードクローン検出を行い，類似したコード片を見つけることで，参照元が記述されてないものに対しても調査を行った．検出されたコード片を URL に基づいた調査と同様，利用状況と再利用後の進化パターンの分類を行った．

結果として OSS に再利用されている Stack Overflow のコード片は 72.4%が最新バージョンのコード片，27.6%が旧バージョンのコード片であった．最新バージョンのコード片が検出されている OSS に対して参照元のコード片の進化に追従しているものを調査したところそのような事例は見られなかった．また，OSS に再利用されている Stack Overflow の旧バージョンが再利用後どのような変更がされているか調査したところ是正保守が 64 件，適応保守が 10 件，完全性保守が 55 件，機能追加が 24 件，非機能が 51 件であり，是正保守が最も多いことが分かった．この中でも是正保守は質問者の実装したい機能を実現させる変更や，バグ修正などでコードが変更されているものがあるため開発者は OSS が再利用している Stack Overflow のコード片の進化を追い，その変更内容を把握して追従すべきである．

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 井上 克郎 教授には，研究において多くの御指導および御助言を賜りました．井上教授の御指導のおかげで，本研究を進めることができ，本論文の完成に至りました．井上 教授に心より感謝いたします．

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，研究の各段階において多くの御助言を賜りました．多くの御指導および御助言を頂いた松下 准教授に深く感謝いたします．

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，研究において大変多くの御指導および御助言を賜りました．多くの御指導および御助言を頂いた神田助教に心より深く感謝いたします．

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 嶋利 一真 氏には，研究に関する相談に乗っていただき，特に，研究調査，論文執筆にあたって数多くの御指導および御助言を賜りました．嶋利 一真 氏に心より感謝いたします．

最後に，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井上研究室の皆様には，多くの御指導及び御助言を頂きました．本論文を完成させることができたことは，井上研究室の皆様の御協力のおかげであると感じております．井上研究室の皆さまに，心より感謝いたします．

参考文献

[1] Hideaki Hata, Christoph Treude, Raula Gaikovina Kula, and Takashi Ishio. 9.6 million links in source code comments: Purpose, evolution, and decay. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, pp. 1211–1221, 2019.

[2] Saraj Singh Manes and Olga Baysal. How often and what stackoverflow posts do developers reference in their github projects? In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR)*, pp. 235–239, 2019.

[3] Saraj Singh Manes and Olga Baysal. Studying the change histories of stack overflow and github snippets. In *Proceedings of the 18th International Conference on Mining Software Repositories (MSR)*, pp. 283–294, may 2021.

[4] StackExchangeInc. Stack overflow. `https://stackoverflow.com/`.

[5] StackExchangeInc. All sites - stack exchange. `https://stackexchange.com/sites?view=list#users`(accessed on 2022-01-29).

[6] Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. Sotorrent: reconstructing and analyzing the evolution of stack overflow posts. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR)*, pp. 319–330, 2018.

[7] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? *Empirical Software Engineering*, Vol. 24, p. 637–673, 2019.

[8] Rabe Abdalkareem, Emad Shihab, and Juergen Rilling. On code reuse from stackoverflow: An exploratory study on android apps. *Information and Software Technology*, Vol. 88, pp. 148–158, 2017.

[9] Stefan Haefliger, Georg von Krogh, and Sebastian Spaeth. Code reuse in open source software. *Management Science*, Vol. 54, No. 1, pp. 180–193, 2008.

[10] Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco. Oliveto. Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering*, Vol. 47, No. 3, pp. 560–581, 2021.

[11] Le An, Ons Mlouki, Foutse Khomh, and Giuliano Antoniol. Stack overflow: A code laundering platform? In *Proceedings of the 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 283–293, 2017.

[12] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy paste on android application security. In *Proceedings of the Symposium on Security and Privacy (SP)*, pp. 121–136, 2017.

[13] Haoxiang Zhang, Shaowei Wang, Tse-Hsun Chen, Ying Zou, and Ahmed E. Hassan. An empirical study of obsolete answers on stack overflow. *IEEE Transactions on Software Engineering*, Vol. 47, No. 4, pp. 850–862, 2021.

[14] Themistoklis Diamantopoulos, Maria-Ioanna Sifaki, and Andreas L. Symeonidis. Towards mining answer edits to extract evolution patterns in stack overflow. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR)*, pp. 215–219, 2019.

[15] Sebastian Baltes and Stephan Diehl. Usage and attribution of stack overflow code snippets in github projects. *Empirical Software Engineering*, Vol. 24, No. 3, pp. 1259–1295, 2019.

[16] F-Droid Limited and Contributors. F-droid - free and open source android app repository. `https://f-droid.org/`.

[17] Toshihiro Kamiya. Aist ccfinderx. `http://www.ccfinder.net/ccfinderxos.html`.

[18] Abram Hindle, Daniel M. German, Michael W. Godfrey, and Richard C. Holt. Automatic classication of large changes into maintenance categories. In *2009 IEEE 17th International Conference on Program Comprehension*, pp. 30–39, 2009.