日本工業大学大学院工学研究科電子情報メディア工学専攻
令和 4 年度修士論文公聴会予稿集

コード片の再利用に向けた依存関係解決と周辺コード補完手法の提案†

上村　勇太*

A Proposal of Dependency Resolution and Peripheral Code Completion Methods for Code Fragment Reuse

Yuta Kamimura

1　はじめに

今日，プログラミング言語には多種多様なものが存在しており，実行前にコンパイルが必要な言語におけるコンパイルエラーの発生とその解決に開発者は多大な時間と労力を費やしている[1].

本研究では Language Server を用いた IDE と依存関係解決ツールを組み合わせる事で，外部ライブラリに依存したコード片を実行可能なツールを開発し，その評価を行なった結果について述べる.

2　目的

本研究の目的は, 開発者が普段から行っているコード片の再利用の作業を支援することである.

ここでは例として, Java で Stream を string に変換したい場面を考える. 1 から書いてもいいが, 面倒なのでコード片を探して使いたい.

```
StringWriter writer = new StringWriter();
IOUtils.copy(inputStream, writer, encoding);
String theString = writer.toString();
```

図 1 依存ライブラリが明記されていないコード片

1. Stack Overflow で「java stream convert string」で検索
2. 一番上に出てきた Q&A[2]を開く
3. ファイルに保存（図 1）
4. class と main がないので囲う
5. 変数宣言と import 文を追加する
6. 「org.apache.commons.io.IOUtils」が外部ライブラリなので必要なライブラリを調べる
7. Apache Commons IO の jar ファイルをダウンロード
8. コンパイル成功

このような単純なコード片であってもこれだけの手作業が必要になってしまう.

コード片がコンパイルできない原因として以下が挙げられる.

1. コード片が Java として正しくない
2. class 及び main 宣言がない
3. import 文がない
4. 未定義のクラス，関数，変数を参照している
5. 外部ライブラリを import している
6. 不明な外部ライブラリに依存している

3　提案手法

本研究では，上述したコード片がコンパイルできない原因を自動的に修正できるツールを提案する.

本手法におけるツールの利用方法を以下に示す.

1. IDE を起動し，ワーキングディレクトリを指定
2. 新規ファイルを作成し，コード片を保存
3. コマンド画面を開き，拡張機能の起動コマンドを入力
4. 補完完了メッセージが表示されるまで待機

これにより，ユーザーは補完済みのコード片とその動作に必要な外部ライブラリを入手することができる.

4　実現手法

本ツールは当該 IDE の拡張機能として実装し，コマンドによって起動するものとした. ツールの利用者は事前に，依存関係解決ツールと IDE 及び, その IDE 上で動作する拡張機能をインストールする必要がある．具体的には，本研究は Java 及び Visual Studio Code[3]を対象としたツー

†本研究の一部は以下において発表した
・ソフトウェア工学の基礎 XXVII 第 27 回ソフトウェア工学の基礎ワークショップ（FOSE2020）
*電子情報メディア工学専攻　2208006　橋浦研究室

ルとしたため，ビルドツールとして Maven[4]，IDE として Visual Studio Code，これに加えて Visual Studio Code 用の Java 言語サポート拡張機能と本ツールをインストールする必要がある．Java 言語サポートには vscode-java[5]を用いる．なお，vscode-java は内部で Eclipse[6]の Language Server[7]を用いている．また，ツール内で構文解析及び抽象構文木の作成に antlr4ts[8]を用いている．

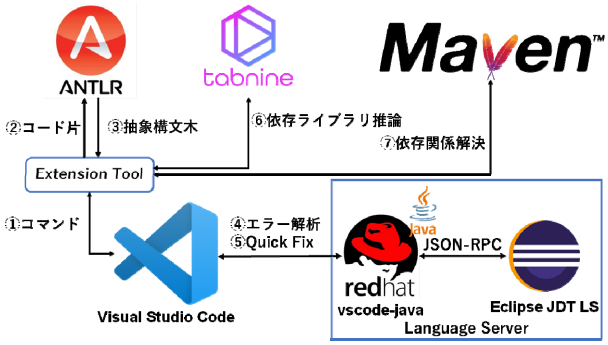

図 2 提案ツールの構成

この拡張機能の基本的な動作の流れは以下の通りである．

1. 補完対象のコード片を開き，コマンドでツールを起動（①）
2. コードが Java の文法違反かどうかを構文解析を用いてチェックし，必要に応じて補完（②，③）
3. 文法違反が存在する場合，補完によってこれが解決できるかをチェックし，解決しない場合は処理を中断
4. 現在のエラー一覧を取得（④）
5. 各エラーの内容を元に，優先順位を付ける
6. 優先順位が最も高い Quick Fix(4.2 節) を 1 つ適用（⑤）
7. 解決可能なエラーが残っている場合，3 に戻る
8. 依存ライブラリの解析，及び解決（⑥，⑦）
9. 解決可能なエラーが残っている場合，3 に戻る
10. 依存ライブラリを含むファイルを生成し，補完が完了したコード片が残る

4.1 文法違反チェックと補完

Java として正しくないコード片でも補完できるようにするため，構文解析を用いた文法違反チェックを行う．

ここでは以下の 2 パターンを自動的に補完する．

- class 宣言の不足
- class と main 宣言の不足

具体的には，対応するスニペットと組み合わせる仕組みを実装した．

なお，ここで言うスニペットとは，コード片の中でも組み合わせる前提のもの（ホットスポットとフローズンスポットがある）とする．

4.2 Quick Fix

現代の IDE は Quick Fix という機能を備えている．これは，IDE が現在発生しているエラーとその修正候補を提示し，開発者は対話式で提示された修正をコードに適用するものである．どのエラーをどのように解決するかは，開発者自らが判断しなければならないが，本研究ではこれをルールベースで連続的に自動で適用することで，不足している import 文の補完処理等を可能にしている．

4.3 依存関係解決機能

本研究では Maven と連携する事で外部ライブラリへの依存関係の解決を行う．

エラーの原因になっているライブラリ名を Maven Central Repository Search の API[9]を用いて検索し，その候補毎に Maven のプロジェクトを生成，ビルドを実行する事で依存関係を解決する．

最後に依存ライブラリを 1 つのファイルにまとめて出力し，コード片のエラーを解消する．

4.4 不明な外部ライブラリの予測

不明な外部ライブラリに依存しているコード片への対応として，本研究ではクラス名やメソッド名から外部ライブラリを予測できる Tabnine[10]を用いる．

Tabnine は Mishne ら[11]の研究を元にしたコード補完サービスであり，機械学習を用いる事でメソッド名やクラス名から依存している外部ライブラリを推測することができる．

5 評価と考察

評価には Stack Overflow の Java タグが付けられた約 150 万個の質問の中から使用頻度の高いキーワード 5 種(csv，dictionary，json，list，xml)で検索したものの内 android タグを除いた vote 上位 10 個に絞り，質問とベストアンサー又は最も評価の高い解答に記載されたコード片 81 個を対象とした．

本研究の研究課題となるのは以下の 4 つである．

RQ1. 補完はどの程度成功したか

RQ2. 補完によって記述量は減少するか

RQ3. 減少した記述はどのようなものか

RQ4. キーワードによって減少した記述はどのように異なるのか

5.1 RQ1:補完はどの程度成功したか

補完の成功率を表 1，表 2 に示す．全体として，コンパイル可能数と補完成功数共に成功率はおおよそ 50%程度であった．

ここで，コンパイル可能とは実行時エラーを考慮せず純粋にコード片がコンパイル可能かどうかを表している．また，補完成功とは，コンパイル可能，かつ，コード片が本来の意図に沿った挙動を示すことが確認できたことを表している．ただし，外部リソース（例えばファイルなど）の読み込みが必要なため，実行時エラーが発生する一部のコード片については，本研究が対象としているコード片の補完自体に問題があるわけではないため補完成功したものと見なすこととした．

表 1 キーワード毎のコンパイル可能率

キーワード	数	割合
csv	10/15	66.6%
dictionary	8/14	57.1%
json	8/20	40.0%
list	12/21	57.1%
xml	6/11	54.5%
	合計 44/81	**平均 54.3%**

表 2 キーワード毎の補完成功率

キーワード	数	割合
csv	9/15	60.0%
dictionary	7/14	50.0%
json	8/20	40.0%
list	8/21	38.0%
xml	3/11	27.2%
	合計 35/81	**平均 43.2%**

5.2 RQ2: 補完によって記述量は減少するか

前節で明らかになったとおり，本ツールは全てのコード片に対してコード補完を行うことはできない. したがって，本ツールを開発に適用する際には，ツールによる自動補完と開発者の手動による補完を併用する必要が生じる．このような場合を想定し，補完を全て手動で行った場合と，ツールを併用した場合を比較して，ツールが目標達成にどの程度寄与しているかについて調査した．

調査方法は, まずコード片が以下の 3 つの状態を考える.

(1).初期状態

(2).ツールによる補完後

(3).補完目標

これらに対して，補完を全て手動で行った場合の記述量(1)→(3)と，ツールを併用した場合の記述量(2)→(3)を求めると, (1)→(2)の部分がツールによって減少した記述量であると定義できる．本研究では diff コマンドによってコードの差分の算出を行った．結果を表 3 及び図 3 に示す.

表 3 キーワード毎のツールの有無での差分行数の平均

キーワード	差分行数の平均	差分行数の平均 (ツール有)	平均の差
csv	9.133	2.266	6.867**
dictionary	12.357	5.786	6.571**
json	3.350	2.650	0.700*
list	13.143	6.905	6.238**
xml	13.727	6.455	7.273**

* $p < 0.05$

** $p < 0.01$

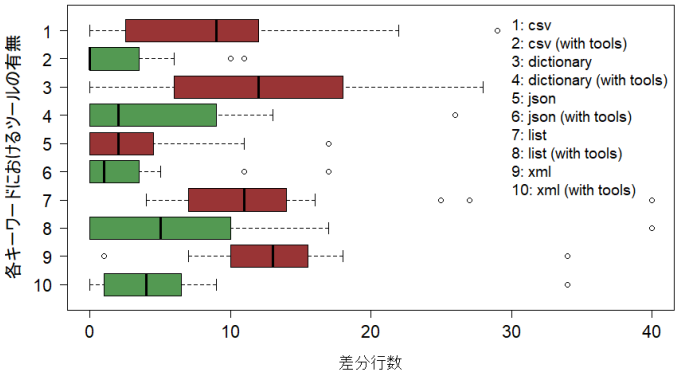

図 3 キーワード毎の差分行数の比較

全てのキーワードにおいて，記述量の減少がみられた．ここで差分行数はツールの有無で変化しないという帰無仮説 H0 を立て，対応のある t 検定を行った結果，json 以外は $p<0.01$, json のみ $p<0.05$ となり帰無仮説が棄却された.

5.3 RQ3: 減少した記述はどのようなものか

5.2 節で述べたツールによって減少した記述について，さらに詳細に調査するために減少した記述について手動で分類を行った．その結果を表 4 に示す.

表 4 キーワード毎の補完成功率

分類	csv	dictionary	json	list	xml
`import` 文	10	9	4	15	8
add `class`	1	1	1	2	1
add `class&main`	9	7	1	13	6
変数宣言	3	3	0	7	4
関数宣言	1	1	0	1	0

表 4 より，import 文の不足や 3.1 節で述べた文法違反チェックとヒューリスティックなスニペット補完に対してツールが有効に作用していることが明らかになった．また，多くのコード片ではこれらの記述が省略されていることが明らかになった.

5.4 RQ4:キーワードによって減少した記述はどのように異なるのか

ここで，キーワード毎によるツールの有効性の違いについて考察を行う．表 4 に示す通り，今回の実験では json に関するコード片については，変数及び関数の宣言を補完したケースが存在しなかった．これは json がデータバインディングに用いられる事が多く，コード片がデータオブジェクトの定義に終始しているパターンが多かったためと考えられる．また，list は変数の宣言の補完が必要である場合が多い．これは頻出の質問がリストの走査や結合であり，操作対象のリストを例示する事が省略される事が多いためである．これらを除くと，本ツールの有効性はキーワードに依存しないことが明らかになった．

6 関連研究

Yang ら[12]は Stack Overflow におけるコード片のパース及びコンパイル可能率を，そのままの場合と必要に応じてスニペットを用いて補完した場合でそれぞれ調査している．Java のコード片の場合，914,974 個のうちパース可能なものは3.89%, コンパイル可能なものは1.00%であった．また，補完後はパース可能なものは 19.24%，コンパイル可能なものは 3.02% まで上昇したと述べている．

Yang らがヒューリスティックな補完に用いたスニペットは class 宣言で囲む 1 種類であったが，本研究ではそれに加えて class と main 宣言で囲む計 2 種類のスニペットを用いている．表 4 の結果が示す通りスニペットによる補完が必要なケースのほとんどは class 宣言で囲むものではなく class と main 宣言で囲むケースであり，この事がパース不能で全く補完処理が行えないパターンの減少に寄与していると考えられる．

Zhang ら[13]は Stack Overflow と GitHub から生成したデータセットを用いて，Stack Overflow に掲載されているコード片からホットスポットを抽象化した状態で提示する Chrome 拡張機能 Example Stack を作成した．これにより，開発者はコードの安全性とロジックのカスタマイズに集中でき，有意に品質が向上すると述べている．

Zhang らの手法はブラウザ上のプルダウンメニューによって手動で加工し，コード片内のロジックを流用するのに特化しているのに対し，本手法では IDE 上でコマンドを入力する事で拡張機能を動作させ自動的に補完することで，コード片に含まれるロジックを改造せずにそのまま動かすことができる．

7 結論

本研究では IDE の機能と依存関係解決ツールを組み合わせたコード片の自動補完ツールを開発し，外部ライブラリに依存したコード片を含むコード片の補完を行った，その結果，コード片に対する周辺コード補完を一定程度自動的に行うことができることが明らかになった，

参 考 文 献

1) Ali Mesbah, Andrew Rice, Emily Johnston, Nick Glorioso, Edward Aftandilian, “Deepdelta: learning to repair compilation errors,” Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 925–936, Aug. 2019.
2) Stack Exchange Inc., “How do I read / convert an InputStream into a String in Java?,” https://stackoverflow.com/search?q=InputStream+read, 2008/11/21, (accessed 2022/07/30).
3) Microsoft, “Visual Studio Code - Code Editing. Redefined,” https://code.visualstudio.com/, (accessed 2022-01-02).
4) The Apache Software Foundation, “Maven - Welcome to Apache Maven,” https://maven.apache.org/, (accessed 2022-01-02).
5) Red Hat Developer, “Language support for Java ™ for Visual Studio Code,” https://github.com/redhat-developer/vscode-java/, (accessed 2022-01-02).
6) Eclipse Foundation, “Eclipse IDE,” http://www.eclipse.org/ide/, (accessed 2022-01-02).
7) Eclipse Foundation, “Eclipse JDT Language Server,” https://github.com/eclipse/eclipse.jdt.ls, (accessed 2022-01-02).
8) Tunnel Vision Laboratories, LLC, “Optimized TypeScript target for ANTLR 4,” https://github.com/tunnelvisionlabs/antlr4ts, (accessed 2022-01-02).
9) Sonatype, Inc., “Maven Central Repository Search,” https://search.maven.org/, (accessed 2022-01-02).
10) TabNine, Inc., “Code Faster with AI Code Completions — Tabnine,” https://www.tabnine.com/, (accessed 2022-01-02).
11) Alon Mishne, Sharon Shoham, Eran Yahav, “Typestate-based semantic code search over partial programs,” OOPSLA, pp. 997–1016, Oct. 2012.
12) Di Yang, Aftab Hussain, Cristina Videira Lopes, “From query to usable code: An analysis of stack overflow code snippets,” 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories, pp. 391–401, May. 2016.
13) Tianyi Zhang, Di Yang, Crista Lopes, Miryung Kim, “Analyzing and supporting adaptation of online code examples,” Proc. of the 41st International Conference on Software Engineering (ICSE ’19), pp. 316–327, May. 2019.

指導教授	審査委員（主査）	准教授	橋浦 弘明
	審査委員（副査）	教授	粂野 文洋
	審査委員（副査）	教授	山地 秀美