Vol.2021-SE-207 No.6
2021/3/1

# Annotation を利用した UML とソースコード間のトレーサビリティリンク作成手法の提案

吉田優介[1]　橋浦弘明[1]　田中昂文[2]　櫨山淳雄[3]　高瀬浩史[1]

**概要**：ソフトウェアの開発や保守において，トレーサビリティリンクは重要である．そのため，トレーサビリティリンクを回復する手法は多く提案されている．しかしながら日本語で書かれた設計書とソースコード間のトレーサビリティリンクを回復する手法はあまり提案されていない．本研究では設計書の要素とソースコードの要素名が一致しない場合でもトレーサビリティリンクを明示的に示す手法を提案する．具体的には Annotation を用いて，対応する設計書の要素名を書くことによってトレーサビリティリンクを回復する．また，この手法をサポートするツールを Eclipse のプラグインとして実装した．本手法が有効であるかどうかを検証するために，比較実験を行った結果について述べる．

**キーワード**：トレーサビリティ，リンク回復，UML，クラス図、Java

# A Proposed Method for Recovering Traceability Links between Documents and Codes Using Annotations

YUSUKE YOSHIDA[†1]　HIROAKI HASHIURA[†1]　TAKAFUMI TANAKA[†2]
ATSUO HAZEYAMA[†3]　HIROSHI TAKASE[†1]

## 1. はじめに

今日，実用的なソフトウェアを開発する場合には，複数人がプロジェクトチームを組み，ウォーターフォールモデルやアジャイル開発プロセスなどを用いて開発を進めていくことが多い. いずれの開発手法を採用していたとしても，開発者は各工程において定められた中間成果物を作成し，後続の工程に引き継ぐ．後続工程の担当者は前工程の中間成果物を元に，次の中間（最終）成果物の作成することで開発を進めていく．

このようなソフトウェア開発や保守において，成果物間のトレーサビリティの確保は重要な問題である．例えば，設計書とプログラムという２つの成果物がある場合を考える．これらのトレーサビリティが確保されていない場合，例えばプログラムには設計書に書かれている機能が実装されていなかったり，設計書と異なった動きをしたりするような矛盾が存在するということになる．このような矛盾は設計書どおりにプログラムが作られていなかったり，プログラムの修正が設計書にフィードバックされていない場合に発生すると考えられるが，そのような経緯に関する情報が失われている場合には，成果物のどちらが誤っているのかを特定することができないという問題も生じる．

このようなトレーサビリティが確保されていない状態の設計書やプログラムに対して機能追加を行おうとする場合，まず，トレーサビリティの回復（成果物間の矛盾を解決）を行ってから機能追加を行わなければならなくなるため，トレーサビリティの回復にかかる分だけコストが増大する．したがって，トレーサビリティが確保されていない状態になることを防ぎながら開発を進めていく必要がある.

ここで，本研究が取り扱うトレーサビリティについて述べる．トレーサビリティとはある工程の中間成果物に含まれている要素が，後続工程で作られた中間成果物の要素と対応が取れているかどうかを表す具体的な要素の対応関係のことをトレーサビリティリンクと呼ぶ.

本研究は，プログラマがコーディングをしている際にトレーサビリティリンクが失われているかを確認できるようにする環境を実現することで，前述のような問題の発生を防ぐ方法を提案する．対象とする設計書はクラス図，ソースコードは java で書かれたものとした.

## 2. 関連研究

UML のモデルとソースコード間のトレーサビリティを取り扱った研究については，これまでも様々なものが提案されている.

まず，代表的なものとして astah*[3]を始めとする UML エディタからスケルトンを生成することができるツールが挙げられる．このようなツールは，モデルからコードのス

1 日本工業大学大学院
Nippon Institute of Technology
2 玉川大学
Tamagawa University
3 東京学芸大学
Tokyo Gakugei University

ケルトンを自動的に生成できるため，トレーサビリティリンクが失われることがない．また，クラス図以外の UML も扱うことができるという利点がある．それに対し本研究で開発したツールは，後続工程側の成果物（ソースコード）に対して追加や修正を行った場合，それが UML 等の設計書に対してフィードバックされているかどうかについても検出することが可能であるという違いがある．

吉川ら[4]は，Model Driven Architecture(MDA)を用いたソフトウェア開発における開発効率の向上を目的としている，MDA を用いたソフトウェア開発におけるモデルとソースコード間の整合性維持ツールを作成している．本研究との違いとしては，対象としている UML がアクティビティ図であること，拡張アクティビティ図の生成などが挙げられる．

伊藤ら[5]は UML 記述の設計モデルとオブジェクト指向言語記述のソースコードを対象としている．クラス名一致やコサイン類似度による対応付けなど，4 つのルールからなるアルゴリズムが自動で抽出したリンク候補から，ユーザーが人手で正しいリンクを選択する半自動的なプロセスをとっている．本研究との違いとして，伊藤らの研究では半自動でリンク付を行っていること，本研究ではアノテーションを使用しているため異なる要素名でもリンク付を可能にしていることが挙げられる．

## 3. 先行研究

著者らは先行研究[1]において UML とソースコード間のトレーサビリティリンク回復手法の提案を行っている．

先行研究では，設計書などは開発者の母国語で記述され，ソースコードには英語が用いられている場合を想定し，英語以外の言語で書かれた設計書と英語で書かれたソースコード間のトレーサビリティリンクの回復を支援する．対象となる成果物は，前工程の中間成果物として UML のクラス図を，後工程の中間成果物としては Java のソースコードとし，トレーサビリティリンクの回復はクラス名，フィールド名，メソッド名の 3 つとした．

前述の内容を実現するために，トレーサビリティリンクを表すための独自アノテーション[2]を定義し，これをソースコード中に埋め込むことによって IDE からトレーサビリティが確保されているかどうかを判断し，利用者にフィードバックを行う．独自アノテーションを利用する利点は以下の 2 点である．

① 書式が統一される
② スペルミスがあった場合にはコンパイラから警告が出るため，記述のミスに気が付きやすい

まず，①に関しては，コメントや Javadoc と比較して，アノテーションは表記方法や挿入位置があらかじめ定められており，開発者の個性による表記のゆらぎを防ぐことができる．

次に②に関しては，アノテーションは付与するターゲットを定義することができるため，定義されたターゲット以外の場所にアノテーションが挿入されている場合，コンパイラから警告を出して開発者に修正を促すことが可能である．

以下にアノテーションを利用した，クラス図（図 1)と Java のソースコード間のトレーサビリティリンクの作成の例を示す．

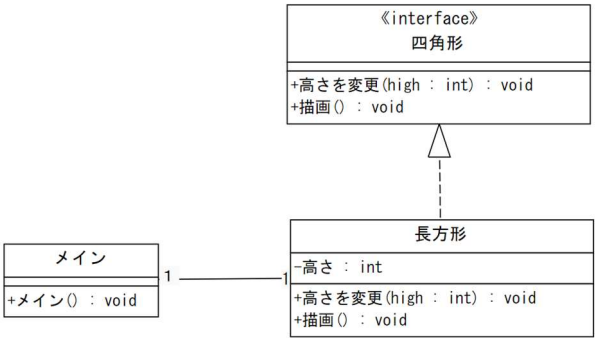


図 1 クラス図の例

本手法ではソースコード中のアノテーションの変数名に対応するクラス図の要素名を記述することで，クラス図の要素とのトレーサビリティリンクを明示的に示す．

具体的には図 1 のクラス図の Interface クラス「四角形」とのトレーサビリティリンクを示す場合，図 2 のように表す．

```
1 @UMLClass(name="四角形")
```

図 2 クラス名のアノテーションの使用例

また，フィールド「高さ」とのリンクを示す場合図 3 のように表す．

```
12    @UMLField(name="高さ")
```

図 3 フィールド名のアノテーションの使用例

さらに，メソッド「描画」とのリンクを示す場合図 4 のように表す．

```
15    @UMLMethod(name="描画")
```

図 4 メソッド名のアノテーションの使用例

先行研究ではこれらの手法により，前述の 3 つの要素に対してトレーサビリティリンクの回復ができることを確認した．その一方で，先行研究ではクラス図に存在する関連をトレーサビリティリンクの回復の対象として取り扱っていなかったため，関連を表現するためのフィールドをソースコード中のクラスに追加すると，その部分が逆にトレーサビリティが取れていない要素としてフィードバックされてしまうという問題が存在し，評価実験の際に問題となっていた[2].

## 4. 提案手法

前述の問題を解決するため，本研究ではクラスの関連とインターフェースの実装をトレーサビリティリンクの回復対象として追加する．

先行研究ではクラスに対して不要なフィールドの追加を防ぐために，依存先のクラスのインスタンスをソースコード中にフィールドとして表現すると，余分なフィールドとしてフィードバックする実装になっていた．

この問題を解決するため，ツールがフィールドの一覧を取得する際，関連を考慮するように拡張し，フィードバックもフィールドの場合とは別に用意した．さらにインターフェースの実装に関しても考慮ができるようにすることとした．

具体的には関連のトレーサビリティリンクを表すためのアノテーションを新たに定義することとした．具体的な関連のアノテーションを図 5 に示す.

```
10  @UMLRelation(name="長方形")
```

図 5 関連のアノテーションの使用例

ソースコード中のフィールドが関連である場合には，このアノテーションを使用する．関連のアノテーションは変数の中にどのクラスのインスタンスを保持しているかを記述する．図 5 の例の場合，クラス「長方形」のインスタンスを保持していることを示している．

## 5. 実現手法

本章ではツールの実現手法について述べる．まず，ツールの全体構成を図 6 に示す.

まず，前工程の中間成果物として UML のクラス図の作成や編集には KIfU[7]を用いる．KIfU は概念モデリングの編集過程のデータを細粒度に収集し，編集過程を明らかにするために開発されたオンラインの UML エディタである．クラスの作成，属性，メソッドの追加，関連の作成など，必要な機能が備わっている．記入されたクラス図の要素はリアルタイムに KIfU のデータベースに格納されるため，本研究で作成するツールはこのデータベースにアクセスすることにより，クラス図の内容を取り出すことができる．

本研究で提案するツールは Eclipse のプラグインとして実装する． 開発者がソースコードを編集すると，データベースからクラス図の情報を取得し，ソースコードの要素との比較をリアルタイムに行い，結果をフィードバックする．このことにより，開発者はクラス図とソースコードを見比べながらトレーサビリティの確保を行うことができる．

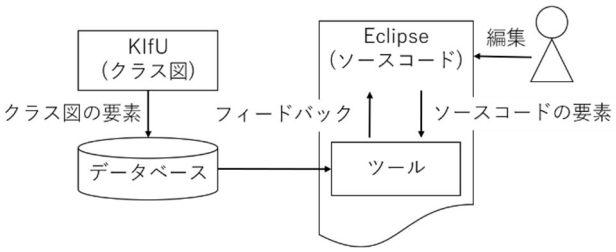


図 6 ツール全体の構成

### 5.1 開発者へのフィードバック方法

開発者へのフィードバックには Eclipse が開発者にエラーを提示するための一般的な方法であるマーカー[8]を用いて実装している．関連に関するフィードバックは大きく分けて 3 種類である．

1 つ目は「余分な関連」である．これはクラス図のクラスの関係が関連になっていないにも関わらずインスタンスをフィールドに保持している場合に表示される．

この場合の具体例を図 7 に示す.

```
@UMLRelation(name="四角形")
余分な関連[四角形]e square = new Square();
```

図 7 余分な関連のフィードバック例

この例では，前工程の成果物が図 1 で示したクラス図である状況において，後工程のソースコードの「メイン」にあたるクラスに対して，「四角形」にあたるクラスのフィールドである square を追加した場合を表している．この場合，「メイン」クラスは四角形のインスタンスを保持するようにクラス図で定義されていないため，余分な関連（クラス図に対応する要素がない）と表示される．開発者はこのようなフィードバックにより，明確にトレーサビリティリンクが失われている要素を認識することができる．このようなエラーに対して開発者は当該のフィールドを削除するか，もしくはクラス図の修正を行うことによってトレーサビリティリンクの回復を図ることができる．

2 つ目のフィードバックは「不足している関連」である．

これは，クラス図のクラスの関係が関連になっているにも関わらず，ソースコードでフィールドがインスタンスを保持していない場合に表示される．

この場合の具体例を図 8 に示す．

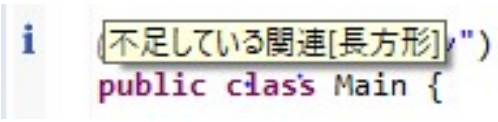


図 8 不足している関連のフィードバック例

この例では前述のフィードバックの例と同様に，前工程の成果物が図 1 で示したクラス図である状況において，ソースコード中の「メイン」にあたるクラスが「長方形」にあたるクラスのインスタンスを保持していない場合を表している．このような場合，クラスの定義の開始部分に必要なフィールドに関するマーカーが表示される．これにより，開発者は不足しているフィールドをソースコードに追加するか，クラス図の修正を行うことによってトレーサビリティリンクの回復を図ることができる．

3 つ目のフィードバックは不適切なインターフェースの実装である．ソースコード中でクラス図に定義されてない実装を行っている場合に「このクラスの継承が間違っています」というフィードバックが表示される．

この場合の具体例を図 9 に示す．

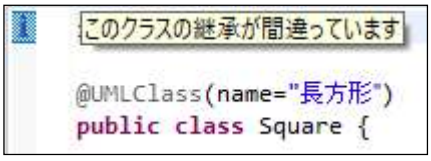


図 9 継承のフィードバック例

この例も前述のフィードバックの例と同様に，前工程の成果物が図 1 で示したクラス図である状況において，「長方形」を表すクラスが「四角形」を表すクラスを実装していることが確認できないことを表している．

このような問題はクラス図では定義されていないインターフェースクラスを実装していたり，クラス図と実装するインターフェースクラスが異なっている場合に発生するだけでなく，インターフェースクラスに適切なアノテーションが付されていない場合にも発生する．開発者はソースコードを修正するか，クラス図の修正を行うことによってトレーサビリティリンクの回復を図ることができる．

本研究が対象とするプログラムは複数のクラスで構成されており，ソースコードの中に表示されるマーカーだけではプロジェクト全体のトレーサビリティリンクの回復ができたかどうかの判別がつきづらいという問題に対応するため，これらのフィールドバックを一覧するための機能を追加した．この機能の具体例について図 10 に示す．

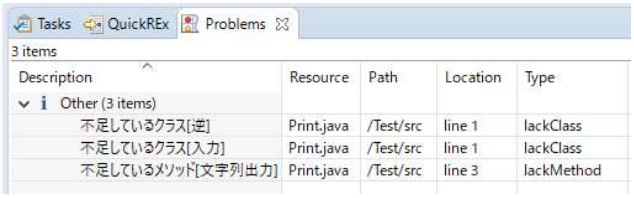


図 10 マーカーの一覧の表示例

## 6. 評価

本手法の有効性を確かめるために．日本工業大学先進工学部情報メディア工学科の学生と日本工業大学大学院電子情報メディア工学専攻の学生，計 5 名に対し実験を行った．実験の目的は本手法によりソフトウェアの機能追加や修正を行う際，どの程度トレーサビリティリンク回復に貢献できるかを明らかにすることである．

実験の概要はあらかじめ用意されたソースコードを拡張し，クラス図と同様，つまりトレーサビリティリンクが回復した状態になるまで修正させるというものである．被験者は 2 つの課題を行う．1 つ目は本稿で提案する手法を用いずに行うものであり，2 つ目は本稿で提案するツールとアノテーションを利用するものである．2 つの問題の難易度をなるべく同等にするため，作問にあたってはクラス数や関連の数については同数になるように配慮を行った．表 1 に問題に含まれる要素数を示す．

表 1 問題ごとの要素数

| | クラス | フィールド | メソッド | 関連 | 依存 | 実装 |
|---|---|---|---|---|---|---|
| **実験 1** | 8 | 5 | 16 | 2 | 4 | 4 |
| **実験 2** | 8 | 5 | 15 | 2 | 4 | 4 |

### 6.1 実験 1: 提案手法を用いない場合

実験 1 では四則演算を行うプログラムを作成する問題である．被験者には図 11 に示すクラス図，対応するソースコード，補足説明を配布した．ソースコードの配布には GitBucket[9]を用いた．対応するソースコードは配布した時点では図 12 のようなクラス構造となっており，加算しか行うことができず，トレーサビリティリンクが失われている状況になっている．被験者はツール等の支援を受けずにトレーサビリティの回復を行う．

### 6.2 実験 2: 提案手法を用いた場合

実験 2 は文字列の変換を行うプログラムを作成する問題である．被験者には図 13 に示すクラス図，ソースコード，補足説明，ツールの仕様方法に関する説明，アノテーションの使用方法に関する説明を配布した．実験 1 の場合と同様にソースコードは配布した時点では図 14 のようなクラス構造となっており，文字列を 2 倍にすることしか行うことができず，トレーサビリティリンクが失われている状況

になっている．被験者はツールの支援を受けながらトレーサビリティの回復を行う．

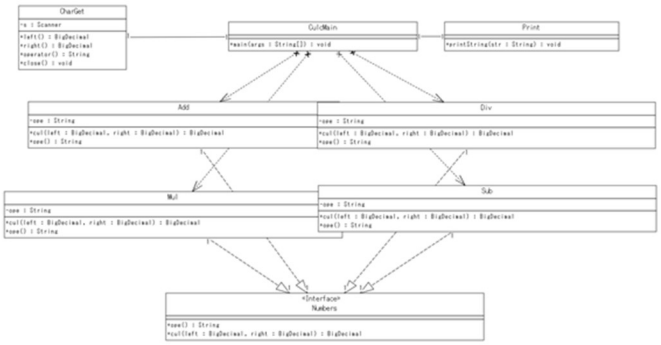

図 11 実験 1 のクラス図

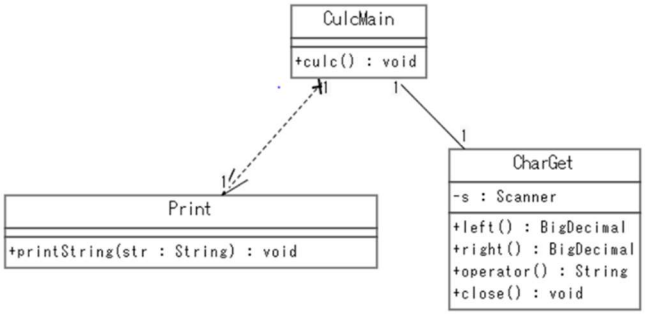


図 12 実験 1 の配布プログラムのクラス構造

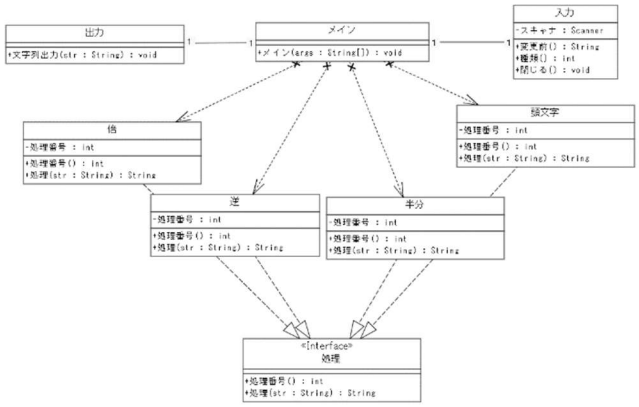

図 13 実験 2 の配布したクラス図

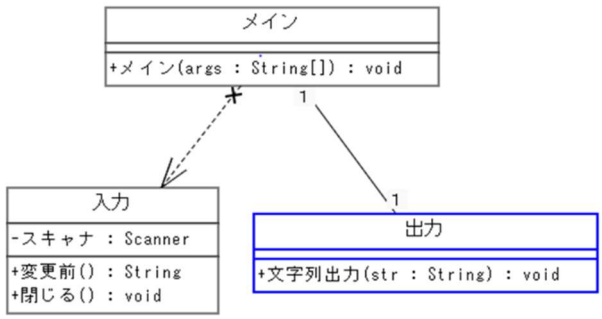


図 14 実験 1 の配布プログラムのクラス構造

### 6.3 評価方法

実験で評価を行う項目は，以下の 5 点である．

① 設計書と比較して，被験者が作成したクラスに過不足がないか
② 設計書と比較して，被験者が作成したフィールドに過不足がないか
③ 設計書と比較して，被験者が作成したメソッドの過不足がないか
④ 設計書と比較して，被験者が作成した関連に過不足がないか
⑤ 設計書と比較して，被験者が作成したクラスのインターフェースの実装に過不足がないか

本研究の目的は設計書とソースコード間のトレーサビリティを確保できるようにすることである．そのため，設計書とソースコードのそれぞれに共通した要素を比較する．ツールが比較する要素がクラス名，フィールド名，メソッド名，関連，実装であるためこの 5 つの要素について評価を行う．

クラス名，フィールド名，メソッド名，関連，実現に関してそれぞれ，適合率(P)と再現率(R)を求める．再現率と適合率の算出方法を以下に示す．

$$P = \frac{\textit{被験者が作成したソースコードの要素のうち，設計書に含まれる要素の数}}{\textit{被験者が作成したソースコードの要素の数}}$$

$$R = \frac{\textit{被験者が作成したソースコードの要素のうち，設計書に含まれる要素の数}}{\textit{設計書に含まれる要素の数}}$$

また，この 2 つの調和平均を評価する．調和平均(F 値)は適合率(P)と再現率(R)を用いて算出する．

## 7. 実験結果と考察

実験結果をそれぞれ表 2，表 3 に示す．
クラス名，メソッド名，実現に関してはツールの補助がなくてもトレーサビリティの回復に差が見られない結果となった．

フィールド名に関してはツールの補助がない場合，トレーサビリティの回復が行えなかった被験者が存在した．具体的には被験者 C は図 15 に示すように 4 つのフィールドが不足していたのに対し，ツールの補助がある場合にはこのような例は皆無であった．

関連についてはツールの効果が最も現れた部分と言える．課題 1 ではソースコードが配布時，クラス「CulcMain」と「Print」の間の関係が依存となっている．そのためクラス図に合わせて修正する場合，フィールドにインスタンスを保持する必要があるが，図 16 に示すように多くの被験

者がこれを実装していない結果となった.

表 2 実験 1（提案手法を用いない場合）の結果

| | | 被験者 A | 被験者 B | 被験者 C | 被験者 D | 被験者 E |
|---|---|---|---|---|---|---|
| クラス | 適合率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 再現率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 調和平均 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| フィールド | 適合率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 再現率 | 1.0 | 1.0 | 0.2 | 1.0 | 1.0 |
| | 調和平均 | 1.0 | 1.0 | 0.3 | 1.0 | 1.0 |
| メソッド | 適合率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 再現率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 調和平均 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 関連 | 適合率 | 1.0 | 1.0 | 0.5 | 0.2 | 1.0 |
| | 再現率 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| | 調和平均 | 0.7 | 0.7 | 0.5 | 0.3 | 0.7 |
| 実現 | 適合率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 再現率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 調和平均 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

表 3 実験 2（提案手法を用いた場合）の結果

| | | 被験者 A | 被験者 B | 被験者 C | 被験者 D | 被験者 E |
|---|---|---|---|---|---|---|
| クラス | 適合率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 再現率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 調和平均 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| フィールド | 適合率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 再現率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 調和平均 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| メソッド | 適合率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 再現率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 調和平均 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 関連 | 適合率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 再現率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 調和平均 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 実現 | 適合率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 再現率 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| | 調和平均 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

```
public class Add implements Numbers{

      @Override
      public String ope(){
          return "+";
      }

      @Override
      public   BigDecimal   cul(BigDecimal   left,
BigDecimal right){
                          .
                          .
```

図 15 フィールドが不足していた被験者のソースコード

```
public class CulcMain{

      private   static   CharGet   charGet   =   new
CharGet();

      public static void main(String[] args){

          Numbers add = new Add();
          Numbers sub = new Sub();
          Numbers mul = new Mul();
          Numbers div = new Div();

          Print print = new Print();
                          .
                          .
                          .
```

図 16 関連を実装していなかった被験者のソースコード

以上の結果から，本提案ツールがトレーサビリティの回復に対して一定の効果があることが確認できたと言える．また，トレーサビリティリンクの回復が十分行えてない部分に着目すると，IDE によるウィザード等が実装されていない部分に集中していることが明らかになった．例えば，今回の実験で用いた Eclipse でサブクラスを作成する場合，この作業をウィザードで実行すると，実装する interface に定義されているメソッドのスタブは自動で生成される機能がある．これにより，クラス図とソースコードのトレーサビリティリンクの回復作業の大半が完了してしまう．このため，ツールの有無による差が出なかったのではないかと考えられる．これに対し，関連についてはそのような機能が IDE に用意されていないため，被験者は本提案ツールによる支援がない場合，完全に自力でトレーサビリティリンクの回復を図る必要があり，トレーサビリティリンクの回復に大きな差が出た可能性がある．

## 8. 結論

本研究ではクラス図とソースコード間のトレーサビリティリンク作成手法の提案を行った．また先行研究で作成した本手法をサポートするツールを改良し，一定の効果を確認した．

### 謝辞

本研究の一部は JSPS 科研費 18K11579 の助成を受けた．

### 参考文献

[1] 吉田優介，橘浦弘明，田中昂文，櫨山淳雄，高瀬浩史，" UML と IDE を利用したモデルとコード間のトレーサビリティ構築支援システムの提案，" 電子情報通信学会 2019 年総合大会 情報・システム講演論文集 2, D-13-2, p.105, Mar. 2019.

[2] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith, “Annotations,” The Java Language Specification, Java SE 11 Edition,

<https://docs.oracle.com/javase/specs/jls/se11/html/jls-9.html#jls-9.7> (accsseed:2021/1/26).

[3] Yusuke Yoshida, Hiroaki Hashiura, Takafumi Tanaka, Atsuo Hazeyama, Hiroshi Takase," A Proposed Method for Recovering Traceability Links between Documents and Codes Written in Different Languages," Proc. of the 2020 RISP International Workshop on Nonlinear Circuits, Communications and Signal Processing (NCSP 2020), pp. 523-526, Feb. 2020.

[4] Change Vision, Inc., "astah*," <https://astah.change-vision.com/ja/index.html> (accsseed:2021/1/24).

[5] 吉川裕基, 片山徹郎, "MDA を用いたソフトウェア開発におけるモデルとソースコード間の整合性維持ツールの試作," ソフトウェア・エンジニアリングシンポジウム 2015 論文集, pp. 145-152, Aug. 2015.

[6] 伊藤弘毅，田邉浩之，波木理恵子，鷲崎弘宜，深澤良彰，"トレーサビリティリンク回復を通じたトレーサビリティ測定と改善支援," コンピュータソフトウェア, Vol. 2013-30, No. 3, pp. 123-129, Jul. 2013.

[7] Takafumi TANAKA, Hiroaki HASHIURA, Atsuo HAZEYAMA, Seiichi KOMIYA, Yuki HIRAI, Keiichi KANEKO, "Learners Self Checking and its Effectiveness in Conceptual Data Modeling Exercises," IEICE TRANSACTIONS on Information and Systems, Vol. E101-D, No.7, pp.1801-1810, Jun. 2018.

[8] Dejan Glozic, "Mark My Words: Using markers to tell users a bout problems and tasks," <http://www.eclipse.org/articles/Article-Mark%20My%20Words/mark-my-words.html> (accessed 2021/1/24).

[9] "GitBucket," <https://github.com/gitbucket/gitbucket> (accessed 2021/1/24).