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 2 年度修士論文公聴会予稿集, pp.69-72

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*

A Proposed Method for Recovering Traceability Links between Documents and Codes that uses Annotation

Yusuke Yoshida

1　はじめに

ソフトウェア開発や保守において、トレーサビリティは重要である。実用的なソフトウェアを開発するためには，ウォーターフォールモデルに代表されるような複数の工程で構成されているソフトウェア開発プロセスが利用されている．開発者は，各工程において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全行程の中間成果物を元に，次の中間成果物の作成を行なう．

本研究におけ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，後続の工程で作られた中間成果物の要素と対応が取れるかどうかという概念を表している．また，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．

トレーサビリティリンクが失われている場合，ソフトウェア開発工程の成果物間に矛盾が存在するということになる．これはソフトウェア開発や保守において大きな問題となる．そのため，トレーサビリティが失われている場合には回復する必要がある。

2　目的

本研究は，プログラマー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ことを目指す．

3　関連研究

本研究の関連研究としてastah*[1]などのUML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．これらのツールは自動で生成するためトレーサビリティリンクが失われることが無い．本研究で開発したツールは機能の追加など，もとからあるソースコードに対して追加や修正を行うことができる．また astah*などはクラス図以外のUML も扱うことができる．

吉川ら[2]の研究は，Model Driven Architecture 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[3]の研究では 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Annotationを使用しているため異なる要素名でもリンク付をできるようにしていることが挙げられる．

4　提案手法

本研究ではクラス図とソースコード間の要素間のトレーサビリティリンク作成手法について提案する．トレーサビリティリンクを作成するために Annotation を利用する．Annotationを利用することで書き方や書かれる場所が統一

†本研究の一部は以下において発表した
・The 2020 RISP International Workshop on Nonlinear Circuits, Communications and Signal Processing (NCSP 2020)

*電子情報メディア工学専攻　2198023　橋浦研究室

される．さらに，スペルミスなど書き間違いをした際に，コンパイラがチェックを行いエラーとして検出できるため，トレーサビリティリンク失われていることにユーザーが気づきやすくなる．Annotation はクラスやフィールドなど，予めターゲットを決めて用意する．こうすることによって他の場所に間違って書くことを防ぐことができる．Annotation の変数に設計書の要素名を入れることで書き換える量を最小限にリンクを作成することができる．さらに，コンパイラが要素名を取得することもできる．

5　トレーサビリティリンク作成手法の提案

トレーサビリティリンクを作成するために独自 Annotation[4]を使用する．Annotaion の変数名に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ことができる(図 1)．

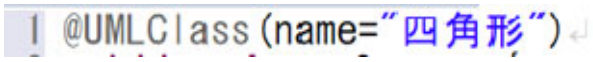

図 1 クラス名の Annotation の使用例

5.1　Annotation を利用する利点

本研究で Annotation を採用した利点は 2 つある．

①書式が統一される

②スペルミスが会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

コメントや Javadoc では書き方や書く位置が統一されないためわかりにくくなる可能性がある．また，Annotation はターゲットが決められているため違うとことに誤って書いた場合にもコンパイラから警告が出るためミスに気が付きやすい．

6　実現手法

ツールは Eclipse のプラグイン形式で実装した．クラス図は KIfU[5]上で編集する．KIfU と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 KIfU のデータベースに格納される．ユーザーがソースコードを編集するとそのたびにデータベースからクラス図の要素を取得し，ソースコードの要素との比較を行う．そのため途中でクラス図を編集した場合でも比較結果に反映される．比較した結果をユーザーにフィードバックする．フィードバック方法は Eclipse のマーカーを用いた．

フィールドを取得する際，フィールドの型名とクラス名の一覧を比較し，該当するものがあった場合にはインスタンスとして扱うように変更した．また継承に関しても取得し比較できるようにした．

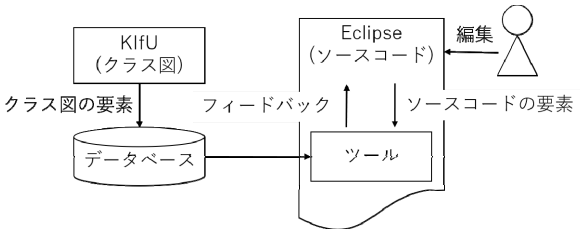

図 2 ツール全体の構成

6.1　フィードバック

フィードバックは 2 種類あり，Eclipse のマーカーに表示される． 1 つは余分な要素，これはクラス図に存在しないがソースコードに存在する場合に表示される(図 3)．

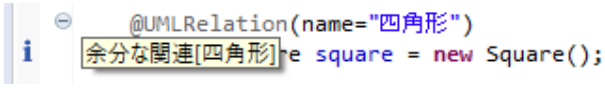

図 3 余分な関連のフィードバック例

2 つ目は不足している要素，これはクラス図には存在しているがソースコードには存在していない要素である(図 4)．

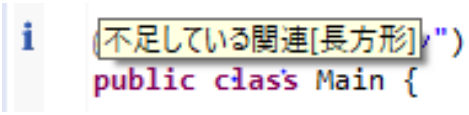

図 4 不足している関連のフィードバック例

これらのマーカーは Eclipse の Problems のタブから一覧で見ることができる(図 5)．

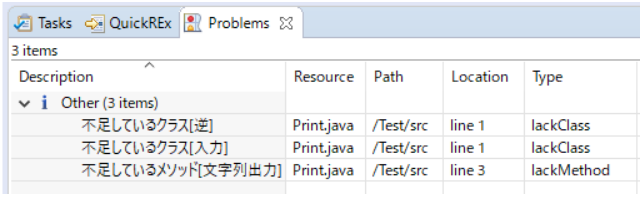

図 5 マーカー一覧の表示例

7　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．

7.1　実験の内容

実験の内容は予め書かれたソースコードをクラス図と同じになるように修正させるという内容である．被験者は 2 つの課題を行った．1 つはツールも Annotation も使用せずにソースコードの修正を行った．2 つ目はツールと Annotation 両方を使用し，ソースコードの修正を行った．

7.2　実験の目的

実験の目的は本手法を使用し，ソフトウェアの機能追加や修正を行う際，本手法がどの程度トレーサビリティリンクの維持に有効であるかを調べることである．

7.3　課題 1

課題 1 では四則演算を行うソースコードを作成した．被験者にはクラス図(図 7), ソースコード, 補足説明を配布した．ソースコードは GitBucket を通して配布した．ソースコードは配布した時点では図 6 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ない．

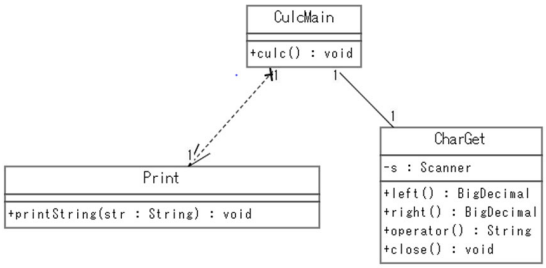

図 6 課題 1 の配布したソースコードのクラス構造

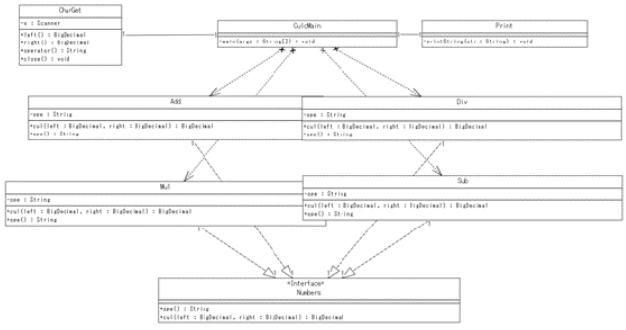

図 7 課題 1 の配布したクラス図

7.4　課題 2

課題 2 では文字列の変換を行うソースコードを作成した．被験者にはクラス図(図 7)，ソースコード，補足説明，ツールの説明，Annotation の説明を配布した．ソースコードは GitBucket を通して配布した．ソースコードは配布した時点では図 6 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ない．

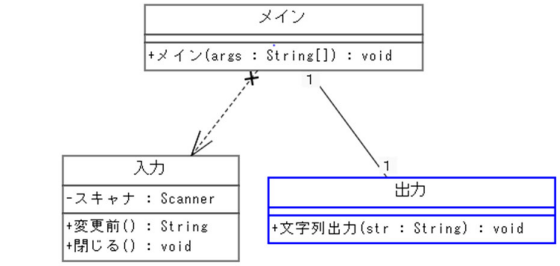

図 8 課題 2 の配布したソースコードのクラス構造

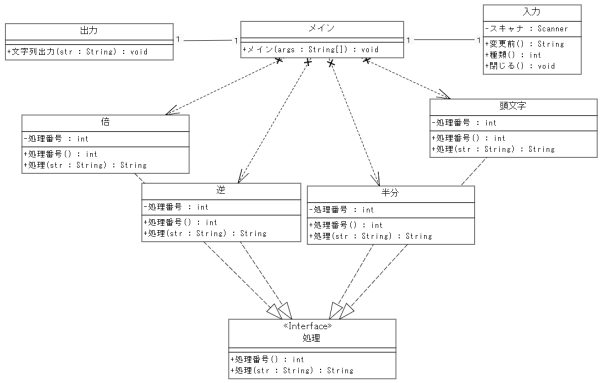

図 9 課題 2 の配布したクラス図

7.5　評価方法

実験を行うにあたって評価する項目は，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継承に過不足がないか

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，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、継承であるためこの 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、継承に関してそれぞれ，適合率(P)と再現率(R)を求める．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．

7.5 実験結果

実験結果は以下の表 1 の通りである．

表 1 実験の結果

		課題 1	課題 2
クラス	平均適合率	1.0	1.0
	平均再現率	1.0	1.0
フィールド	平均適合率	1.0	1.0
	平均再現率	0.8	1.0
メソッド	平均適合率	1.0	1.0
	平均再現率	1.0	1.0
関連	平均適合率	0.7	1.0
	平均再現率	0.5	1.0
継承	平均適合率	1.0	1.0
	平均再現率	1.0	1.0

クラスに関してはツールの補助がなくてもミスが起きなかった．フィールドに関してはツールの補助がない場合にクラス図と差異が生じた被験者がいた．被験者 C は継承している 4 つのクラスのフィールドが不足していた．ツール使用時にはフィールドの不足は解消されていた．メソッドに関してはツールの補助がなくてもミスが起きなかった．関連はツールの効果が最も見られた．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多くの被験者がこれを実装していなかった．ツールを使用した課題 2 ではすべての被験者がこれを修正していた．継承に関してはツールの補助がない場合にもミスが起きなかった．

8 考察

実験結果から元ある部分を変更する場合にミスが生じやすいのではないかと考えた．被験者 C の場合，継承しているクラスのフィールドを実装していなかった．Eclipse で継承クラスを作成する場合，interface で実装されているメソッドは自動で生成される．そのため，ある程度完成しているものに書き加える必要がある．このような場合ひと目見ただけでは気が付きにくいためミスが生じやすいのではないかと考えた．同様に，多くの被験者が行っていなかった依存になっている部分を関連に変更する作業ももとからある部分を変更する作業であるため気が付かなったと考えた．

9 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．また本手法をサポートするツールを作成し，効果を確認した．

参 考 文 献

1) astah*, https://astah.change-vision.com/ja/index.html (accsseed:2021/1/24).
2) 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, pp145-152, 2015-08-31.
3) 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.
4) インターフェース Annotation, https://docs.oracle.com/javase/jp/8/docs/api/java/lang/annotation/Annotation.html (accsseed:2021/1/26).
5) 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.

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	高瀬 浩史
	審査委員（副査）	教授	粂野 文洋