意図しない設計変更を防ぐラウンドトリップエンジニアリング支援手法†

山崎　貴弘*

A Method for Supporting Round-Trip Engineering with the Ability to Avoid Unintended Design Changes

Takahiro Yamazaki

1　はじめに

Scrum[1]などの短期間で成果物を作るアジャイル開発手法は成果物に関する顧客との認識の齟齬を減少させる手法であるこれらの手法を用いるには，設計とコーディングを何度も往復しながら開発を行う（ラウンドトリップエンジニアリング)[2] 必要がある．ラウンドトリップエンジニアリングとは，システム開発中の成果物の一貫性を保つのに開発と局所的な修正を繰り返す作業である [1]．ラウンドトリップエンジニアリングの利点として以下 2 つがあげられる．1 つ目に，開発者が 1 人で修正を完結することによりコストを抑える．2 つ目に，プロトタイプ開発などの反復的な開発手法で有効である．その一方で，前述のような設計とコーディングの往復を繰り返すと成果物間で一貫性・整合性が失われる問題が生じ次のような問題が生じる．1．設計書の規模が大きくなるにつれ，開発者による一貫性・整合性の維持が困難になる[3]．２．修正の記録や内容が修正担当者のみに蓄積され，その内容を他のメンバーが把握できないことにより，開発や保守が困難になる[4].

2　目的

この問題を解決するために本論文では，プログラムから設計成果物へのトレーサビリティリンク手法を提案し，それを支援するツールを開発した．本手法の有効性を調べるために以下の評価を行う．

1. 半自動による修正作業において，実装上の理由などでクラス図に変更を加える必要がある場合と，プログラマが開発メンバーに相談せずにコードに変更を加える場合で修正の受け入れに差が出るかを明らかにする
2. ある要件を満たすためにソースコードの変更をクラス図に反映させる作業を行う場合，　完全自動化ツールと本手法(半自動化)の修正率を比較する．

3　提案手法

本研究は，プログラムから設計成果物へのトレーサビリティリンクの作成手法を提案する．本手法では設計成果物としてクラス図，ソースコードは Java を扱う．

著者らは，本研究で扱うラウンドトリップエンジニアリングにおいて，開発者がクラス図とソースコード間の一貫性・整合性が失われる問題が発生する原因として，開発者の目視チェックによる成果物間のずれの見逃しとラウンドトリップエンジニアリング中の細かな仕様変更の 2 点に着目した．前者は例えば開発者が"Scanner"クラスの"e"を"a"と入力ミスしたのを見逃した場合に発生する．"Scannar"は英単語としては正しくないが，プログラム内でこのような名前が一貫して用いられていればプログラムの実行に不具合は生じないので，このような間違いは見逃されやすい．

後者は開発者がプログラミング中に設計の誤りを見つけて修正を行った場合に発生する問題である．具体例を挙げると，設計の段階で入力を専門とするクラス"Input"を作成し，プログラミングの段階になってからこのクラスを入出力クラスとして作成した方が良いことに気づいてプログラムの仕様を変更し，プログラム上のクラス名も実態に合わせて"InOut"に変更したような場合に発生する．このような場合，プログラムを優先して設計を変更すべきか，設計を優先してあくまで設計時の構造を維持したプログラムに修正すべきかについては高度な判断を要する問題となる．提案手法ではこれらの 2 点を踏まえ，クラス図とソースコード間の一貫性・整合性を維持する方法として，ツールによって修正案の候補を提示し，開発者がその中から選択をするという半自動修正のアプローチを採用することとした．

クラス図に対する修正候補の提示のイメージを図 1 に示す．本研究で扱うクラス図とソースコード間の整合性の問題については，それらの間に差分がある状態とし，その状態に応じて以下の 3 つに分けて取り扱う．

---

†本研究の一部は以下において発表した
・電子情報通信学会知能ソフトウェア工学研究会(2022 年 1 月)

*電子情報メディア工学専攻　2218012　橋浦研究室

A) 余剰差分：クラス図上にはあってソースコード上にはないもの
B) 欠損差分：ソースコード上にあってクラス図上にはないもの
C) 置換差分：要素名は同じだが修飾子または，型が異なる要素

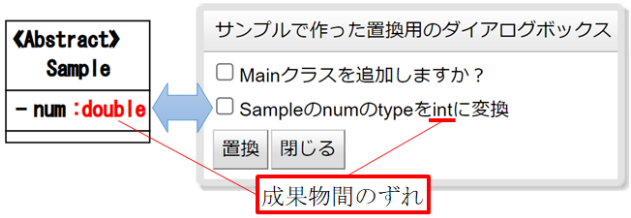

図 1 ツールによる差分表示の例

3.1 差分の提示方法

前節で述べた差分の提示方法について述べる．本手法では差分はクラス図上に赤文字で開発者に提示する．クラス図に示すことができない欠損差分はダイアログ上での提示となる．具体的な提示方法を図 1 に示す．トレーサビリティの問題リンクの表示方法差分を提示する要素は以下の 5 つとした．

i. ステレオタイプ：Interface, Abstract
ii. クラス名
iii. フィールド：修飾子，要素名，型
iv. メソッド：修飾子，要素名，戻り値
v. パラメータ：パラメータ名，型

3.2 修正履歴の保存

置換ボタンが選択されると，ダイアログの状態を保存する処理が実行される．ダイアログの内容を保存する理由は，修正履歴を保存しておくことで，開発者間で同様の問題が発生した場合に，変更の判断基準を再利用することが可能となるからである．保存される修正履歴は以下の通りである．開発者が閉じるボタンを押した場合は，情報をデータベースに保存せずにダイアログを閉じる．

1) ユーザーがどのような選択をしたか(チェックボックスの値)
2) 差分内容の詳細．ダイアログ上に表示される文章が保存される
3) ダイアログが開かれたとき，閉じられた時のタイムスタンプ
4) 開発者が修正の判断をした理由をテキストで保存．

4 実現手法

本手法はソースコードの記述に Eclipse, モデリングツールには KIfU[5]を用いる．本手法のツールは Eclipse のプラグイン部と KIfU の拡張部の 2 つで構成される．ツールによる処理の概要を図 2 に示す．

ツールの具体的な処理手順を以下に述べる．

1) Eclipse プラグインはソースコードに変更が加わったことを検知すると，ソースコードから要素を 取得し，データベースへ格納する．
2) Eclipse プラグインは 1)が完了すると JeroMQ[6]を用いて KIfU に通知する．
3) KIfU は 2)の通知を受信すると，1)でデータベースに格納されたソースコードと KIfU で編集中のクラス図の差分を検出する．
4) KIfU は 3)で検出した差分個所をクラス図上に表示する(図 1)．
5) KIfU は差分に応じた修正候補の一覧(図 1)を開発者に提示する．開発者は適切なものをチェックボ ックスで選択することにより，クラス図の自動修 正を行うことができる．

ツールが提示する修正候補の一覧は，要素名に基づき，3 章で述べた成果物間の差分の種類別に以下の ルールに従って生成する．

a) 余剰差分：クラス図からの余剰要素の削除
b) 欠損差分：クラス図への欠損要素の追加
c) 置換差分：修飾子や型をソースコードに合わせて変更

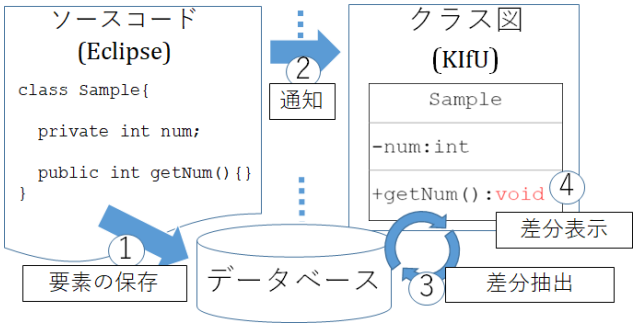

図 2 ツールの実装図

5 評価と考察

本手法が意図しない変更を防げるかを調べるため，評価実験を行った．実験では，クラス図とソースコードの間に不整合がある課題の修正率（挿入された不具合を用意した要求通りに解消できた率）を 2 つで比較した．比較実験 1. 被験者は，プログラムに変更を加えた理由がある問題と理由がない状態の両方の問題を手作業で修正する．比較実験 2. 自動修正ツールとマニュアルでの修正率の比較．参加者は，日本工業大学先進工学部情報メディア工学科学部生 5 名，電子情報メディア工学専攻に所属する 4 名の合計 9 名である．

表 1 マニュアルにおいて与えた状況が違う時の実験結果

	正解率		適合率		再現率		F 値	
	問題 1 根拠有	問題 2 根拠なし	問題 1 根拠有	問題 2 根拠なし	問題 1 根拠有	問題 2 根拠なし	問題 1 根拠有	問題 2 根拠なし
余剰	68.3%	59.7%	47.1%	31.0%	88.9%	50.0%	61.5%	38.3%
欠損	56.9%	51.4%	36.2%	31.1%	94.4%	77.8%	52.3%	44.4%
置換	63.9%	44.4%	38.2%	21.1%	72.2%	44.4%	50.0%	28.6%
フィールド	63.0%	48.1%	37.0%	22.7%	94.4%	55.6%	53.1%	32.3%
メソッド	70.8%	54.3%	44.4%	27.9%	66.7%	66.7%	53.3%	39.3%
パラメータ	62.2%	55.6%	51.6%	45.5%	88.9%	55.6%	65.3%	50.0%

表 2 本手法と自動修正ツールの実験結果

	正解率		適合率		再現率		F 値	
	問題 1 手作業	自動	問題 1 手作業	自動	問題 1 手作業	自動	問題 1 手作業	自動
余剰	68.3%	28.6%	47.1%	28.6%	88.9%	100.0%	61.5%	44.4%
欠損	56.9%	25.0%	36.2%	25.0%	94.4%	100.0%	52.3%	40.0%
置換	63.9%	25.0%	38.2%	25.0%	72.2%	100.0%	50.0%	40.0%
フィールド	63.0%	22.2%	37.0%	22.2%	94.4%	100.0%	53.1%	36.4%
メソッド	63.0%	23.5%	33.3%	23.5%	66.7%	100.0%	44.4%	38.1%
パラメータ	62.2%	40.0%	51.6%	40.0%	88.9%	100.0%	65.3%	57.1%

比較実験 1 の手順は以下の通りである．まず，参加者にはクラス図と Java のソースコードからなる課題と各問題の要求書が与えられる．これは問題 1 と 2 で共通の部分である．問題 1 と問題 2 で異なる点として問題 1 ではコードに変更が加わった理由が与えられ，問題 2 ではコードに変更が加わった理由は与えられない．問題 1 は，プログラマーはプログラミング上の都合で成果物に変更を加えたい理由があり，その変更内容が要求を満たすかどうかを議論する場面．問題 2 は，プログラマーがメンバー間と相談をせずに勝手にプログラムに変更を加えた場面を想定している．

比較実験 2 では，自動修正ツールがソースコードをリバースエンジニアリングし，変更内容をクラス図に反映した時の修正率と比較実験 1 の結果を比較するものである．問題は比較実験 1 と同じものを使用し，自動修正ツールには IBM Rhapsody[7]を使用した．

実験の評価を行うにあたって RQ を 2 つ立てた

RQ1：マニュアルの中でも与えた状況が違う時にどのような違いが現れるのか．

RQ2：自動修正ツールとマニュアルでの修正作業にはどのような特徴があるのか．

5.1 RQ1：与えた状況に違いがある場合のマニュアル修正の特徴

半自動の修正作業において実装上の都合がある時とない場合での修正率の違いを調査した．特徴的な結果として表 1 欠損の F 値は，実装上の都合があろうとなかろうと修正の判断に違いが出ないことが明らかになった．これは，設計時に入念に議論が行われていればその設計成果物には不足している物はないと考えるのが普通であり，実装者の都合があったとしても，設計通りに作って欲しいと思うのが設計者の心情だからだと考える．例として，リストのソートを行うクラスがあった時に　実装者はソートを行うためにインスタンスフィールドである tmp を追加したいと考えたとする．しかし，設計者の考えでは，ソートのアルゴリズムを自作するのではなく標準ライブラリの sort()などを利用すれば余分な変数を追加する必要はないと想定しており新しい要素は追加しないリジェクトの判断になりやすい．よって，実装上の都合は欠損要素に対して影響を与えないと考えた．

対して表 1 余剰の F 値はどの項目においても差が大きい．参加者は"実装の段階で使用しなかった" などの実装上の都合を確認し，その後の保守作業でシステム全体像を理解しやすくするために開発中に使用しなかった変数と関数を削除する方が効果的であると判断したと考えた．

5.2 RQ2：自動修正ツールとマニュアル

半自動と全自動ツールの比較において，表 2 の再現率と適合率から自動修正ツールは見逃しがないが，受け入れる必要がない欠陥を受け入れてしまうことを定量的に示した．対して半自動修正では F 値の値が自動修正ツールに勝って

いるが値はそこまで高くはない．ここから実装上の都合により必要な変更を探し出し受け入れることができるが，与えられた実装上の都合が悪い方向にも働くと考えられる．

6 関連研究

Yu ら[8]は，設計成果物とソースコード間の IR ベースのトレーサビリティ確保に着目している．彼らは，自然言語で説明される設計成果物とプログラミング言語とで語彙の不一致があり，トレーサビリティの精度に影響を与えることを問題点として挙げている．それに対して　彼らは，2 つの成果物間に共通しているデータベースの Statement を IR 技術と組み合わせる手法を提案している．彼らの手法はトレーサビリティ確保の実験において IR 手法の 1 つである VSM より Precision と F2Measure の値が高いことが示されている．

Yoshida ら[9]は，設計からプログラムへのトレーサビリティリンクを作成する手法を提案している．彼らは UML とソースコード間で差異のある要素名を Java の Annotation 機能を用いてトレーサビリティリンクを作成しており，他の研究と比較して日本語と英語などの異なる要素名でトレーサビリティリンクを作成することができるようにしている．しかしながら，彼らは設計成果物からプログラムへのトレーサビリティリンクの作成のみに留まっており，プログラムから設計成果物へのトレーサビリティリンクを作成する機能を備えていないため，ラウンドトリップエンジニアリングにツールを用いることができない．

Jongeling ら[10]は，モデルベース開発のラウンドトリップにおけるモデル・ソースコードの同期を提案している．特に，産業オートメーション領域では，手動で修正したソースコードのモデルへの逆伝播が課題となっている．彼らが提案したツールは，ソースコードのどこが変更されたかを特定し，モデルとソースコードの差分を XML に出力することで開発者に変更箇所を提示するものである．彼らの研究とは対照的に本研究には，モデリングツール上で直接差分を表示するため，モデルとソースコードの比較に必要な時間と労力を削減することができる利点がある．

7 結論

アジャイル開発手法のような往復型のエンジニアリングプロジェクトでは，成果物間の整合性を保つことが困難である．この問題を解決するためには本論文では，プログラムから設計成果物へのトレーサビリティリンク手法を提案し，それを支援するツールを開発した．

半自動の修正作業において実装上の都合がある時とない時での修正率の違いを調査した．特徴的な結果として欠損の値は，実装上の都合があろうとなかろうと修正の判断に違いが出ないことが明らかになった．これは，設計者にとって設計成果物は要求通りに設計が完了しているという前提にあり，あとから要素を追加しなくて開発を進めてほしいという心情にあるのだと考える．半自動と全自動ツールの比較において，本手法は自動修正ツールよりも整合性を維持することができることを定量的に示した．自動修正ツールは見逃しがないが余計に変更を加える特徴がある．

参 考 文 献

1) K. Schwaber, “Scrum development process: Advanced development methods,” in OOP-
2) SLA’95 Workshop on Business Object Design and Implementation, Oct. 1995, pp. 117–134.
3) Tsuchiya, R., Nishikawa, K., Washizaki, H., Fukazawa, Y., Shinohara, Y., Oshima, K., and Mibe, R. , “Recovering transitive traceability links among various software artifacts for developers,” IEICE Transactions on Information and Systems, vol.E102D(9), pp.1750-1760, 2019.
4) J. Guo, N. Monaikul, and J. Cleland-Huang, “Trace links explained: An automated approach for generating rationales,” in 2015 IEEE 23rd International Requirements Engineering Conference (RE), 2015, pp. 202–207.
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, 2018.
6) Trevorbernard, “Java-ZeroMQ,” Dec. 2021, https://zeromq.org/languages/java/
7) IBM Corporation, “IBM Engineering Systems Design Rhapsody 9.0.1,” 2020, https://www.ibm.com/products/systems-design-rhapsody
8) L. Yu, Y. Li, Y. Feng, and C. Qi, “Traceability method between design documents and source codes based on sql dependency,” in 2021 20th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES), 2021, pp. 144–147.
9) Y. Yoshida, H. Hashiura, T. Tanaka, A. Hazeyama, and H. Takase, “A proposed method for recovering traceability linksbetween documents and codes written in different languages,” in The RISP International Workshop on Nonlinear Circuits, Communications and Signal Processing 2020 (NCSP 20), Feb. 2020, pp. 1–4.
10) R. Jongeling, S. Bhatambrekar, A. Lofberg, A. Cicchetti, F. Ciccozzi, and J. Carlson, “Identifying manual changes to generated code: Experiences from the industrial automation domain,” in 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2021, pp. 35–45.

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	高瀬　浩史