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

伊藤 弘毅　田邉 浩之　波木 理恵子　鷲崎 弘宜　深澤 良彰

トレーサビリティはソフトウェアの保守性を左右する一つの要素であり，これまで様々な管理手法が報告されてきた．しかし，現状ではトレーサビリティの達成度合を示す指標が提供されていないため，その認識は主観的なものになりがちである．我々は，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間におけるトレーサビリティの定量的測定手法を提案する．具体的には，独自の半自動アルゴリズムにより設計と実装を対応付け，GQM 法（Goal-Question-Metric approach）で定めた枠組みを利用し測定を実施する．提案手法を実際の設計モデルとソースコードの組に適用した結果，第三者でも少ない労力でトレーサビリティを把握できることを確認した．

As traceability is a factor in software maintainability, various methods have been reported to preserve it. However, perception of traceability tends to be subjective because there is no indicator which represents the degree of it. We propose an approach to measure traceability between a design model described in UML and source code written in object-oriented language. In detail, our approach maps a design to its implementation with the semi-automatic original algorithm and measure traceability with the Goal-Question-Metric approach. We apply the approach to pairs of a design model and source code and find that the degree of traceability can be presumed with a little work even by a third person.

1　はじめに

トレーサビリティ―開発プロセスにより生成された複数プロダクト間の関連度合[8]―はソフトウェアの保守性における重要な要素である．その有用性として，設計方針決定支援[3] やドキュメントの理解[5]，インパクトアナリシス[4] などが報告されてきている．

トレーサビリティの有用性を活かすには，その正しい認識が不可欠である．例えば，片方のみを変更したことで内容が相違した設計モデルとソースコードに対して，新たな機能を追加することを考える．この際，相違に対する対応として以下のことが考えられる．

1. 保守作業の前に設計情報を変えることにより，両者の相違を解消する
2. 相違度合が非常に大きい場合，ソースコードからリバースして設計モデルを作成し直す
3. 保守終了まで一時的に相違への対処を見送る

上記の中から状況に適した行動を選択するには，具体的な相違箇所の情報のほかに，全体的なトレーサビリティの認識が必要となる．しかし，相違情報そのものは客観的な観点を与えないため，その認識は主観的になりがちである．もし，プロジェクト内でトレーサビリティに対する共通認識がなければ，組織として保守の方針を定めることが困難となってしまう．

我々は 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間のトレーサビリティ測定手法を提案する．また，プロジェクトでリンクを維持していない場合，測定前にそれを人手で回復するには非常にコストがかかる．そこで我々は，設計と実装を対

Traceability Measurement and Improvement via Recovering Traceability Links
Hiroki Itoh, Yoshiaki Fukazawa, 早稲田大学, Waseda University.
Hiroyuki Tanabe, Rieko Namiki, 株式会社オージス総研, OGIS-RI Co., Ltd..
Hironori Washizaki, 早稲田大学，国立情報学研究所 GRACE センター, Waseda University, National Institute of Informatics (GRACE Center).
コンピュータソフトウェア, Vol.29, No.1 (2012), pp.78–84.
[研究論文 (レター)] 2011 年 12 月 14 日受付.

応付けるアルゴリズムについても提案し，その有用性について議論する．本論文の研究課題を以下に示す．

RQ1 設計モデルとソースコード間において必要なリンクを回復できるか

RQ2 リンクの回復を省コストで実現できるか

RQ3 抽出したリンクに基づいて，トレーサビリティを測定できるか

RQ4 トレーサビリティの測定結果に基づいて，その改善を支援できるか

そして，上記研究課題に対する本論文の貢献は以下の通りである．

- 設計モデルとソースコード間におけるリンクの回復手法の提案
- 設計モデルとソースコード間におけるトレーサビリティの測定枠組みと修正方針の提案
- リンク回復・トレーサビリティ測定・相違箇所の視覚化を実現するツールの実装

2 トレーサビリティ測定手法の提案

トレーサビリティの測定は，図 1 に示す 4 ステップから成る．ユーザーは 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入力し，相違の視覚化結果とトレーサビリティ測定結果を得る．

2.1 設計モデルとソースコードの対応付け

プロジェクト内でリンクが管理されていない場合，測定の前にそれを回復する必要があるため，我々は 4 つのルールから成る対応付けアルゴリズムを定義した．本アルゴリズムは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る．これは，全自動による完全なリンク回復が難しく，誤ったリンクは正確な測定を妨げるためである．図 2 に我々が作成したツールにおける選択画面を示す．なお，各ルールで複数のクラスが対応する場合はその全てを候補として提示する．

Rule 1 と 2 は名称一致によりクラスを対応付ける．

Rule 1: クラスパス一致の対応付け

設計モデルとソースコード中のクラスのうち，クラス名（名前空間を考慮）が一致するものを対応付ける．

Rule 2: クラス名一致の対応付け

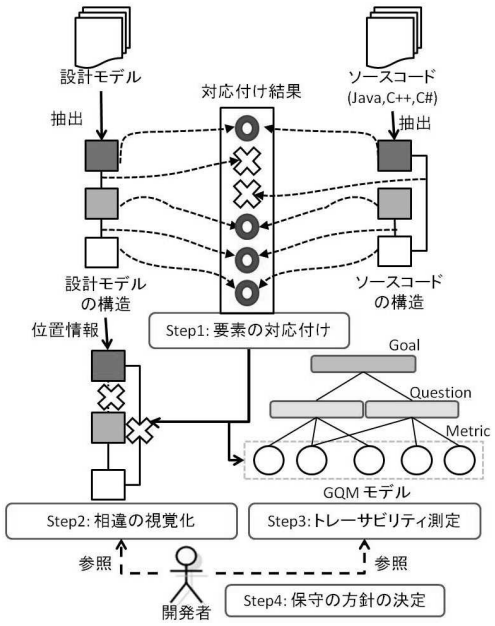

図 1 トレーサビリティ測定の全体像

図 2 ユーザーによる正解リンクの選択画面

Rule 1 で対応付いていない設計モデルとソースコード中のクラスのうち，クラス名（名前空間を考慮しない）が一致するものを対応付ける．

しかし初期の設計モデルの場合，洗練による構造変化が想定されるため，上記のルールだけでは不十分である．この問題に対処するのが Rule 3 と 4 である．

Rule 3 はクラス名のコサイン類似度の値により対応付けをするものである．これは追加クラスが，既存のクラスの名前に類似した名前を持つ可能性を考慮して定義した．また Rule 4 は，クラス分割などにより追加されたクラスを検出することを目的に，テキスト

情報と構造的な特徴を総合してクラスを対応付ける．

Rule 3: コサイン類似度を用いた対応付け

以下の 2 つの条件を満たすクラス（設計モデル中：C_d，ソースコード中：C_i）を対応付ける．

1. C_i は Rule 1 と 2 により対応付いていない．
2. C_d の名称が持つ語彙と C_i の名称が持つ語彙のコサイン類似度が閾値 T_3 以上

Rule 4: 関連・汎化を伴う抽出に関する対応付け

以下の 3 つの条件を満たすクラス（設計モデル中：C_d，ソースコード中：C_i）を対応付ける．

1. C_i は Rule 1 と 2 により対応付いていない
2. C_i は，既に C_d と対応付いている設計中のクラスの誘導可能な関連端または汎化の関係である
3. C_d と C_i の Vocabulary Coverage Metric (VCM) の値が閾値 T_4 以上

VCM の定義は以下の通りである．V_d は設計モデル中の対象クラスに含まれる語彙の集合，V_i はソースコード中の対象クラスに含まれる語彙の集合とする．

$$VCM(V_d, V_i) = \frac{|V_d \cap V_i|}{|V_i|}$$

2.2 相違箇所の視覚化

我々はモデル間の相違を 3 種類に分類し，色やステレオタイプで識別した図を提示するツールを実装した．それぞれの相違の定義は以下の通りである：

- 追加：設計モデルに記述されていないソースコード中の要素
- 除去：ソースコードに記述されていない設計モデル中の要素
- 修正：設計モデルにもソースコードにも記述されているが，内容が異なる要素

ツールが出力する相違の視覚化例を図 3 に示す．

2.3 トレーサビリティ測定の枠組み

我々は，2 つの視点に基づきトレーサビリティを測定する．一つはシステム視点でのトレーサビリティであり，設計モデルとソースコードに共通して存在するクラスやクラス間関係の数などを評価する．もう一つはクラス視点でのトレーサビリティであり，両者に共通して存在する属性や操作の数などを評価する．

我々は，上記 2 つの視点に基づくトレーサビリティの把握をゴールとし，GQM 法[2] を適用した．表 1 にシステム視点でのトレーサビリティ，表 2 にクラス視点でのトレーサビリティに対する適用結果を示す．なお，それぞれの Question は設計が実装要素を網羅しているか，およびその逆に着目するよう設定した．

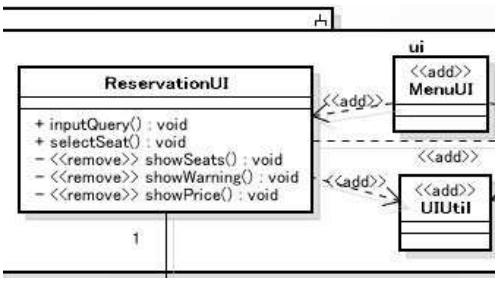

図 3 相違箇所の視覚化出力例

全ての Metric を測定したあと，Goal と Question の得点を算出する．これらの得点はそれぞれの Goal と Question が持つ Metric の値の平均値とする．

3 測定結果の利用

本章では，トレーサビリティの測定結果を保守の方針の決定に適用する方法を記述する．保守の方針を決定する上で，Question の得点が重要な役割を果たす．

システム視点でのトレーサビリティは，対象成果物が適切な開発プロセスにより作成されたものかを示す指標となっている．表 3 はこの視点における 2 つの Question（S-Q1，S-Q2）の関係をまとめたものである．S-Q1 の得点が低い場合は，設計意図が正しく実装に反映されていないことを示唆しているため，設計モデルの破棄，再構築を考慮したほうがよい（表中の III）．また，S-Q2 の得点はソースコードに新しい機能を追加すると低くなる傾向があるため，得点が低く設計意図にない追加機能がある場合，モデルやソースコードの修正を考慮すべきである（表中の II）．

クラス視点でのトレーサビリティは，各クラスが持つ情報が妥当かどうかを示唆している．表 4 に C-Q1 と C-Q2 の得点に応じた修正方針をまとめた．この測定結果と対象クラスの重要性に応じて優先順位をつけることで，修正作業の効果的な実施が可能となる．

表 1 システム視点でのトレーサビリティに対する GQM 法の適用結果

Goal	Question	Metric
S-G. システム視点での設計モデルとソースコードのトレーサビリティ	S-Q1. 設計モデルを構成するクラスやクラス間の関係はソースコードに対応付けられるか	S-M1. 設計モデルに記述されているクラスのうち，ソースコードに対応付けられるものの割合 (%)
		S-M2. 設計モデルとソースコードで置かれている名前空間が一致するクラスの割合 (%)
		S-M3. 設計モデルに記述されているクラス間関係のうち，ソースコードに対応付けられるものの割合 (%)
	S-Q2. ソースコードで実装されているクラスやクラス間の関係は設計モデルに対応付けられるか	S-M4. 設計モデルに対応付けることができるソースコード中のクラスの割合 (%)
		S-M2. 設計モデルとソースコードで置かれている名前空間が一致するクラスの割合 (%)
		S-M5. 設計モデルに対応付けることができるソースコード中のクラス間関係の割合 (%)

表 2 クラス視点でのトレーサビリティに対する GQM 法の適用結果

Goal	Question	Metric
C-G. クラス視点での設計モデルとソースコードのトレーサビリティ	C-Q1. 設計モデル中に記述されているクラスの内容やクラス間関係はソースコード中に記述されているか	C-M1. 設計モデルに記述されている属性のうち，ソースコードに対応付けられるものの割合 (%)
		C-M2. 設計モデルに記述されている操作のうち，ソースコードに対応付けられるものの割合 (%)
		C-M3. （対応付けられる属性・操作のうち）可視性・型が一致する属性，可視性・返り値の型・引数が一致する操作の割合 (%)
		C-M4. 設計モデルに記述されているクラス間関係のうち，ソースコードに対応付けられるものの割合 (%)
	C-Q2. ソースコード中に実装されているクラスの内容やクラス間関係は設計モデルに記述されているか	C-M5. ソースコード中に実装されている属性のうち，設計モデルに対応付けられるものの割合 (%)
		C-M6. ソースコード中に実装されている操作のうち，設計モデルに対応付けられるものの割合 (%)
		C-M3. （対応付けられる属性・操作のうち）可視性・型が一致する属性，可視性・返り値の型・引数が一致する操作の割合 (%)
		C-M7. ソースコード中に実装されているクラス間関係のうち，設計モデルに対応付けられるものの割合 (%)

表 3 S-Q1 と S-Q2 の得点が示唆する対応方針

		S-Q2 の得点	
		高	低
S-Q1 の得点	高	I. 良好なトレーサビリティ	II. 追加クラスに関する相違の修正を検討
	低	III. 設計モデルの破棄・再構築を検討	

表 4 C-Q1 と C-Q2 の得点が示唆する対応方針

		C-Q2 の得点	
		高	低
C-Q1 の得点	高	i. 良好なトレーサビリティ	ii. ソースコードのクラスの修正や全体的な構造の変更を検討
	低	iii. 対象クラスの正当性について検討	

4 評価実験

4.1 実験内容と結果

(1) 対応付け手法の効率性に関する実験

表 5 に示す 8 プロジェクトに対応付け手法を適用し，リンク候補を抽出，その中から正しいリンクを選択した．ただし，$T_3 = 0.75$，$T_4 = 0.4$ と設定した．表 5 にリンク候補の総数と正解数を，表 6 に各ルールに対するリンク候補の情報を示す．c は各ルールにより抽出されたリンク候補，s は c の中から筆者が正しいと判断したリンクを意味する．また，p は筆者が実際に設計モデルとソースコードを見比べて設定したリンク，適合率は#s/#c，再現率は#s/#p である．

(2) トレーサビリティ測定結果の有用性に関する実験

プロジェクト P3 と P4 を対象にトレーサビリティ

表 5　プロジェクト規模と対応付け結果

Project	設計モデル		ソースコード			Total				
	作成段階	#class	言語	#class	LOC	#p	#c	#s	適合率	再現率
P1	初期段階	31	C++	54	2703	43	49	40	81.6%	93.0%
P2	初期段階	23	C++	29	2148	16	16	16	100.0%	100.0%
P3	後期段階	22	C++	26	1497	21	22	21	95.5%	100.0%
P4	後期段階	31	C++	32	2239	30	31	30	96.8%	100.0%
P5	初期段階	31	Java	139	8399	82	91	73	80.2%	89.0%
P6	後期段階	25	Java	28	3554	24	25	24	96.0%	100.0%
P7	初期段階	11	Java	13	319	13	11	11	100.0%	84.6%
P8	初期段階	19	Java	23	478	16	16	16	100.0%	100.0%

表 6　対応付けアルゴリズムの評価

Project	Rule 1			Rule 2			Rule 3			Rule 4		
	#c	#s	適合率	#c	#s	適合率	#c	#s	適合率	#c	#s	適合率
P1	28	28	100.0%	1	1	100.0%	10	10	100.0%	9	1	11.1%
P2	0	0	—	14	14	100.0%	2	2	100.0%	0	0	—
P3	21	21	100.0%	0	0	—	1	0	0.0%	0	0	—
P4	30	30	100.0%	0	0	—	0	0	—	1	0	0.0%
P5	0	0	—	55	55	100.0%	29	14	48.3%	19	16	84.2%
P6	0	0	—	24	23	95.8%	1	1	100.0%	0	0	—
P7	10	10	100.0%	0	0	—	1	1	100.0%	1	1	100.0%
P8	16	16	100.0%	0	0	—	0	0	—	0	0	—

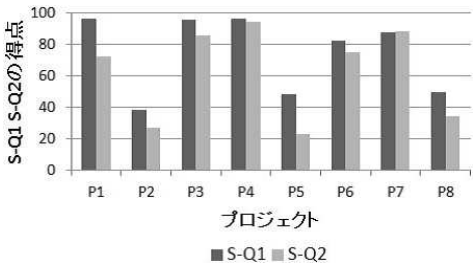

図 4　8 プロジェクトに対する S-Q1 と S-Q2 の得点

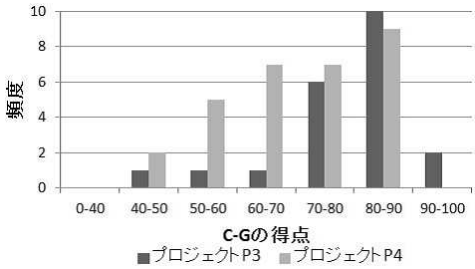

図 5　プロジェクト P3 と P4 に対する C-G の得点分布

測定結果からその状態と修正方針を評価した．図 4 にプロジェクトごとの S-Q1 と S-Q2 の得点を，図 5 に各クラスの C-G の得点分布を示す．

プロジェクト P3 に対するシステム視点でのトレーサビリティの得点は図 4 の通り高水準であり，クラス視点についても 2 つのクラスの得点（47.0 点と 52.4 点）以外は高得点だった．これより我々は，P3 はトレーサビリティが十分に保たれており（表 3 内の I），低得点の 2 クラスのみ後に修正すべきと判断した（表 4 内の ii）．一方，P4 に対するシステム視点での得点は 8 プロジェクト中最高であったが，クラス視点の得点は多くが低い値を示した．これより我々は，P4 は全体的な構造を保てているが，クラスが持つ情報を表 4 内の ii の指針に従い修正の必要があると判断した．

測定後，上記の評価の妥当性を検証するため，トレーサビリティの実情を知る開発者とのインタビューを実施した．その結果，プロジェクト P3 ではトレーサビリティを意識して開発をしており，実際にその維持を積極的にしていたことが判明した．一方，P4 における実際のトレーサビリティはおおむね評価結果と

一致していたが，いくつかのクラス視点の得点が実際よりも低いとの印象を開発者に与えた．調査の結果，その原因は記述が省略された属性や操作が，実装で新しく追加された要素と誤認識されたためと分かった．

4.2 研究課題に対する考察

RQ1 設計モデルとソースコード間において必要なリンクを回復できるか

表 6 から，Rule 1 と 2 は全てのプロジェクトについて正確かつ多くのリンク候補を導出できることが分かる．対照的に，Rule 3 と 4 が導出したリンク候補は比較的少数で，一部は低い適合率を示している．

ただし誤ったリンク候補を破棄することは簡単だが，新しくリンクを作るには対応付けるクラスの選択に時間がかかるため，適合率よりも再現率の値を評価すべきである．表 5 によると，5 つのプロジェクトは全ての，他の 3 プロジェクトは約 9 割の正解リンクが候補として提示されたことが分かる．これより，提案手法は正解リンクの大部分を抽出し，ユーザーが新規に作成するリンクの数を減らせることを示している．

RQ2 リンクの抽出を省コストで実現できるか

被験者にプロジェクト P3 のリンクを手動で，P4 のリンクをツールで回復させる実験をした．なお，被験者は P3 の開発者である．実験の結果，手動のリンク回復に 14 分 41 秒，ツールによる回復に 5 分 19 秒要した．クラス数が多く前提知識がない P4 の方が少ない時間で回復されており，ツールの使用でリンク回復にかかる時間を大きく減らせることが確認できた．

RQ3 抽出したリンクに基づいて，トレーサビリティを測定できるか

4.1 のインタビュー結果から，プロジェクト P3 と P4 の測定値はトレーサビリティをおおむね妥当に反映していると思われる．ただし P4 のクラス視点の得点に関する事例から，評価する際には記述の省略が正確な測定を妨げることを留意する必要がある．

RQ4 トレーサビリティの測定結果に基づいて，その改善を支援できるか

インタビューの結果，プロジェクト P3 と P4 双方の修正方針について賛同が得られた．また P3 では，低得点であった 2 クラスを開発者が問題と認識できていなかったことから，測定結果が予期せぬ相違の検出に有用であることも確認できた．この提案方針とツールから出力された相違の視覚化情報の利用により，効率よい修正作業の実施が可能になると思われる．

5 関連研究

リンクの回復に関しては，以下の既存研究がある．Gethers らは複数の情報検索手法を統合し，要求—ソースコード間のリンク回復手法を提案した[7]．また，Dagenais らは API とドキュメント間のリンク回復手法を提案している[6]．しかし，設計と実装間に特化した回復手法は Antoniol らの手法[1] しか存在せず，それは実装段階での構造変化を考慮していない．

6 おわりに

我々は設計モデルとソースコード間のトレーサビリティ測定手法を提案した．これにより，少ない労力でトレーサビリティや修正方針を把握できるようになる．展望として，より大規模で様々な対象について評価実験を実施し，新たな知見を得たいと考えている．

参 考 文 献

[1] Antoniol, G., Caprile, B., Potrich, A. and Tonella, P. : Design-code traceability for object-oriented systems, *Annals of Software Engineering*, 9(1-2), 2000, pp. 35-58.

[2] Basili, V. R., Cladiera, G. and Rombach, H. D. : The goal question metric approach, *Encyclopedia of Software Engineering*, John Wiley & Sons, Inc., 1994, pp. 528-532.

[3] Brcina, R., Riebisch, M. : Defining a Traceability Link Semantics for Design Decision Support, *ECMDA-TW '08*, 2008, pp. 39-48.

[4] Briand, L.C., Labiche, Y., O'Sullivan, L. : Impact Analysis and Change Management of UML Models, *ICSM '03*, 2003, pp.256-265.

[5] Campos, P., Nunes, N.J. : Practitioner Tools and Workstyles for User-Interface Design, *IEEE Softw.*, 24(1), 2007, pp. 73-80.

[6] Dagenais, B. and Robillard, M. : Recovering Traceability Links between an API and Its Learning Resources, *ICSE '12*, 2012, pp. 47-57.

[7] Gethers, M., Oliveto, R., Poshyvanyk, D. and Lucia, A. D. : On Integrating Orthogonal Information Retrieval Methods to Improve Traceability Recovery, *ICSM '11*, 2011, pp. 133-142.

[8] IEEE : *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.