一般社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

派生プロダクト群における要求・実装間のトレーサビリティリンク抽出

土屋 良介[†]　鷲崎 弘宜[†]　深澤 良彰[†]
加藤 正恭[‡]　川上 真澄[‡]　吉村 健太郎[‡]

† 早稲田大学情報理工学科 〒169-8555 東京都新宿区大久保 3-4-1
‡ 日立製作所横浜研究所 〒244-0817 神奈川県横浜市戸塚区吉田町 292 番地
E-mail: ryousuke_t@asagi.waseda.jp

あらまし 派生開発されてきたプロダクト群に対して，保守や拡張を行う際には，要求と実装の間でトレーサビリティが明確化されていることが望ましい．しかし，大規模なプロダクト群に対して，これらの作業を人手のみで行うことは困難である．我々は，構成管理ログを用いて，要求資産とソースコード，それぞれの共通性・可変性分析結果間の対応関係を分析し，その結果を利用することで，要求・実装間のトレーサビリティリンクを自動的に抽出する手法を提案する．提案手法を，一定規模の派生プロダクト群に適用した結果，実用的な時間内で，妥当なトレーサビリティリンクを抽出できた．

キーワード トレーサビリティ，派生プロダクト，構成管理ログ，共通性・可変性分析

Extraction traceability links between requirements and implementation in the same series of software products

Ryosuke Tsuchiya[†]　Hironori Washizaki[†]　Yoshiaki Fukazawa[†]
Tadahisa Kato[‡]　Masumi Kawakami[‡]　Kentaro Yoshimura[‡]

† Dept. Computer Science, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan
‡ Yokohama Research Laboratory, Hitachi, Ltd 292 Yoshida-cho, Totsuka-ku, Yokohama-shi, Kanagawa, 244-0817 Japan
E-mail: ryousuke_t@asagi.waseda.jp

Abstract When performing maintenance and expansion for the group of products derived, it is important to clarify traceability between requirements and implementation. However, for the large group of products, these tasks manually are difficult. First, we analyze commonality and variability in requirements and source codes. Next, using the configuration management log, we analyze the correspondence of those results. Finally, using the results of this study, we propose a method to extract traceability links between requirements and implements automatically. Result of applying the proposed method to derived products group of a certain size, we were able to extract the appropriate traceability links in a practical time.

Keyword Traceability, Derived Products, Configuration Management Log, Commonality and Variability Analysis

1. はじめに

派生開発とは，既存のソフトウェアを基に，機能の追加・変更等を行うことで，派生的にソフトウェアを開発する手法である．派生開発されてきたプロダクト群に対して，保守や拡張を行う際には，要求と実装の間でトレーサビリティが明確化されていることが望ましい．トレーサビリティとは，追跡可能性のことであり，ソフトウェア開発においては，情報と情報の関係を追跡できることを指している．また，その具体的な情報間の関係や繋がりを，トレーサビリティリンクと呼ぶ．

要求・実装間のトレーサビリティが曖昧であると，開発効率は著しく低下する．例えば，要求の変更があった場合，要求と対応する実装箇所が分かっていないと，すぐに実装作業に移ることができない．

また，派生開発を繰り返すことにより，トレーサビリティが低下するケースも存在する．リリース間隔が短くなることで，仕様書等の文書の管理が適切に行われないケースである．例えば，新機能の実装，または既存機能の削除等を行った際，要求仕様書の更新が行われていないと，要求・実装間で正しい対応関係が築けなくなる．

上述の通り，派生開発を行う上でトレーサビリティは重要である．個々の要求，実装箇所の間に明確なトレーサビリティリンクが形成されていることが理想である．しかし，実際の派生プロダクト群においては，ソフトウェア規模の増大，リリース間隔の短縮などの要因によって，明確なトレーサビリティリンクの形成は十分に行われていない．

トレーサビリティが明確である必要があるのは，最新のプロダクトに限った話ではない．最新版では取り除かれた機能であっても，新たに開発するプロダクトに再利用する可能性もある．したがって，派生開発されてきたプロダクト群全ての要求・実装間のトレーサビリティリンクが必要となる．しかし，個々のプロダクト毎にトレーサビリティリンクを抽出していては，複数プロダクトに共通な機能の存在により，作業の重複が起こる．

This article is a technical report without peer review, and its polished and/or extended version may be published elsewhere.
Copyright ©2012 by IEICE

派生プロダクト群には，全てのプロダクトに共通な機能がある一方で，特定のプロダクト固有な機能がある．それぞれの機能がどのプロダクトに属しているかを調べ，プロダクト群における共通部分・可変部分を分析することを，共通性・可変性分析と定義する．

派生プロダクト群における要求資産とソースコードについて共通性・可変性分析を行い，共通部と可変部を明確に分ける．それによって，要求・実装間のトレーサビリティリンク抽出に重複作業がなくなる．

2. 研究課題

以上で述べた背景を基に本研究の目的を述べ，目的を達成する為の研究課題を定義する．

大規模な派生プロダクト群のトレーサビリティリンクの抽出を，人手のみで行うことは現実的でない．そこで我々は，大規模な派生プロダクト群における要求・実装間のトレーサビリティリンクを自動的に抽出することによって，派生開発の効率を向上させることを目的とする．

本研究の目的を達成する上での課題は，大規模な派生プロダクト群における要求・実装間のトレーサビリティリンクを，実用的な時間で自動的に抽出可能な手法の確立である．

要求と実装の抽象度には大きな差があり，同一の概念を異なる形で表すことが多い．したがって，トレーサビリティリンクを抽出する為には，要求を表す語と実装を表す語の対応関係を示した情報が必要となる．

そこで我々は，要求と関連のある情報と，実装箇所の情報が含まれる構成管理ログに注目した．構成管理ログとは，ソフトウェアに対して行われる構成管理の記録のことである．本研究では，ソフトウェア開発に際して，開発者が記述したメッセージと，追加・修正を行ったファイルのパスが，それぞれ記録されたリビジョンの集合を構成管理ログとして扱う．

構成管理ログから得られる情報を利用して，要求・実装間のトレーサビリティリンクを抽出する．以下に，本研究の研究課題を定義する．

RQ1 構成管理ログを持つ大規模な派生プロダクト群において，要求・実装間のトレーサビリティリンクを自動的に抽出できるか

RQ2 対象となる派生プロダクト群が大規模であっても，実用的な時間内に，要求・実装間のトレーサビリティリンクを抽出できるか

以上の 2 つの研究課題が，提案手法により解決可能か検証を行っていく．

以下，本論文では，まず 3 章で，提案手法について説明する．次に 4 章で，一定規模の派生プロダクト群に行った提案手法の適用実験について報告する．5 章で，関連研究との比較を行い，6 章で，まとめと今後の課題について述べる．

3. 提案手法

3.1. 全体像

本研究の適用対象は，同一組織内で作られた大規模な派生プロダクト群とする．対象が，派生プロダクト群における全ての要求資産とソースコードになるので，プロダクト群の共通性・可変性分析を行ってから，トレーサビリティリンクの抽出を行う．したがって，提案手法を以下の 3 つのステップに分けて設計した．

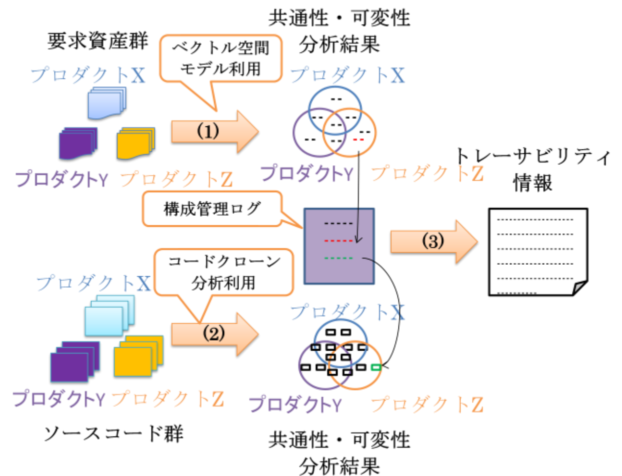

図 1 提案手法の全体像

(1) ベクトル空間モデルを用いた要求資産の共通性・可変性分析

(2) コードクローン分析を用いたソースコードの共通性・可変性分析

(3) 構成管理ログを用いた要求・実装間のトレーサビリティリンクの抽出

以上の 3 つを含めた，提案手法の全体像を図 1 に示す．

準備段階として，ステップ(1)，(2)で，派生プロダクト群における要求資産・ソースコードの共通性・可変性分析を行う．分析手法に利用する要素技術としては，既存研究でも利用されているベクトル空間モデルとコードクローン分析を用いる．

そして，ステップ(3)で，要求資産・ソースコードそれぞれの共通性・可変性分析結果を対象に，構成管理ログを利用することで，要求と実装箇所の対応関係を抽出する．その対応関係を分析することで，要求・実装間のトレーサビリティリンクを抽出する．

成果物として，要求資産・ソースコードの共通性・可変性分析結果，要求・実装間のトレーサビリティ情報が出力される．以下の節で，3 つのステップについて，それぞれ説明する．

3.2. 要求資産の共通性・可変性分析

このステップでは，派生プロダクト群における，要求資産の共通性・可変性分析を行う．本研究における要求資産とは，プロダクトに対する要求を，自然言語で記述した文書群を指す．ただし，分析を自動的に行う為に，全ての要求を一行ずつ羅列した文書を，入力フォーマットとする．

同一ドメインのプロダクト群における要求資産の共通性・可変性分析手法を，熊木らが提案している[1]．ベクトル空間モデルにおける文書間類似度を用いて，異なるプロダクト間の要求同士が，同一のものであるか否かを判断する．我々は，派生プロダクト群の要求資産の共通性・可変性分析に，この手法を利用する．

ベクトル空間モデルとは，Salton らによって提案された，文書内の出現単語に基づいて，文書を 1 つのベクトルとして表現し，要素となる単語の出現数により

決定されるベクトルの向きを元に，文書の内容を判断する技法である[2].

異なるプロダクト間の要求を全て比較し，文書間類似度を計算した結果，それぞれの要求は，全てのプロダクトに共通な要求，一部のプロダクトにしか存在しない要求に分類される．要求がどのプロダクトに属するかの分類結果は，全てのプロダクト群を一つの集合とした際の部分集合として表現される．

3.3. ソースコードの共通性・可変性分析

このステップでは，派生プロダクト群における，ソースコードの共通性･可変性分析を行う．本研究では，分析対象の最小単位を関数とする．それぞれのプロダクトのソースコードにおける関数･コンポーネントが，プロダクト群において，全てのプロダクトに共通する共通部であるか，一部のプロダクトにのみ存在する可変部であるかを分析する．本研究では，1 つのファイルや 1 つのクラスを，コンポーネントと呼ぶ．

コードクローン分析を用いて，ソフトウェア間の類似度の分析や，ソースコード群を対象とした共通性・可変性分析を行う研究がある[3][4]．コードクローンとは，ソースコード中に存在する同一または類似なコード断片のことである．異なるソフトウェア間を跨って存在するコードクローンを検知・分析することで，ソフトウェア同士の類似度を分析することができる．我々も，派生プロダクト群のソースコードの共通性・可変性分析に，コードクローン分析を利用した．以下で，本研究での分析法を簡単に説明する．

まず，それぞれのプロダクトのソースコードから関数群を抽出する．抽出された関数群に対して，コードクローン分析を行い，異なるプロダクト間を跨ってクローンを共有する関数同士を，グルーピングする．

次に，抽出されたグループを対象とし，グループを構成する関数の間の類似度を測定する．それぞれの関数がもつコードクローンの字句数をT_i，それぞれの関数の総字句数をE_iと表すと，N個の関数における，関数間の類似度Sを，本研究では以下の式で定義する．

$$S = \frac{\sum_{i=1}^{N} T_i}{\sum_{i=1}^{N} E_i}$$

ユーザーの設定した閾値以上の類似度を持つ関数グループでは，グループ内の関数全てが同一の関数であると判断される．閾値以下の場合は，異なる関数と判断され，グルーピングは解かれる．

コードクローン分析により，関数のグルーピングと類似度計算を行った結果，それぞれの関数が，プロダクト群における共通部・可変部に分類される．

関数レベルの共通性・可変性分析を行った後，コンポーネントレベルの共通性・可変性分析を行う．関数群の分析の際に抽出された関数グループに着目し，グループ内のそれぞれの関数が属するコンポーネント同士を，類似コンポーネントグループとしてグルーピングする．コンポーネント同士の類似度を計算する際は，コードクローンの字句数，総字句数共に，コンポーネントが持つ関数群の字句数を合計したものを利用する．

コンポーネントレベルでも，関数レベル同様，閾値によるフィルタリングを行い，共通部と可変部に分類する．関数・コンポーネントがどのプロダクトに属するかの分類結果は，要求と同様，全てのプロダクト群を一つの集合とした際の部分集合として表現される．

3.4. トレーサビリティリンク抽出

3.4.1. トレーサビリティリンク抽出方法

このステップでは，先のステップで得た要求資産の共通性・可変性分析結果と，ソースコードの共通性・可変性分析結果の間の対応関係を分析する．そして，分析結果を基に，対象となる派生プロダクト群における要求･実装間のトレーサビリティリンクを抽出する．

我々は，構成管理ログのメッセージ内に，要求と関連する言葉が記述されていると考えた．そこで，それらの言葉を利用することで，要求と実装箇所の対応を得る手法を設計した．

まず，1 つ 1 つの要求について，それらを特徴付けるキーワードを複数設定する．キーワードは，要求仕様書等のドキュメントから自動的に抽出するか，ユーザーが直接指定する．

対応関係の抽出は，ソースコードのドメイン毎に行う．本研究では，同一機能を実現するコンポーネントの集合をドメインと定義する．例として，コンポーネントをまとめるディレクトリが挙げられる．

構成管理ログにおいて，対象ドメインのソースコードのパスを含むリビジョン群に対し，要求毎に設定したキーワード群を用いて検索する．一定数以上かつ一定種類数以上（ユーザー指定）のキーワードが，メッセージ内に表れているリビジョンを特定し，そのリビジョンに記されているパスが示すコンポーネントと，要求を対応付ける．

得られた対応関係の情報には，2 種類の情報が付加される．1 つ目は，抽出元リビジョンの分類であり，2 つ目は，要求・実装それぞれが属するプロダクトの組み合わせを照らし合わせた結果である．これらの情報を元に，要求・実装間のトレーサビリティリンクを抽出する．以下の節で，付加情報について説明する．

3.4.2. 抽出元リビジョンの分類

我々は，構成管理ログのリビジョンを，以下の 2 つのタイプに分類した．

Type A. 修正・追加されたコンポーネント群が，単一のドメインに属するリビジョン

Type B. 修正・追加されたコンポーネント群が，複数種のドメインに分かれているリビジョン

この分類の意図は，対応関係の分解能を区別することにある．Type A のリビジョンからは，要求・実装の対応が 1 対 1 や多対 1 のトレーサビリティリンクを抽出することができる．一方で，Type B のリビジョンからは，多対多の対応は抽出できるが，そこから 1 対 1，多対 1 の対応を自動的に抽出することは困難である．後者のリビジョンから得られた対応関係は，分解能の低い情報として抽出する．その情報から，ユーザーが 1 対 1，多対 1 のトレーサビリティリンクを見出す．

3.4.3. プロダクトの組み合わせの照合

ある要求と，コンポーネントの対応関係が抽出された際，それぞれが属するプロダクトの組み合わせは，直感的には合致するはずである．しかし，実際は様々な要因により，必ずしも合致するとは限らない．例えば，全てのプロダクトに共通する要求と，初期のプロダクト固有のコンポーネントが対応付けられていた場

合，原因としては，要求仕様書の更新漏れや，要求の粒度が大きすぎるといったものが考えられる．このような，仕様書の問題の情報や，要求・実装の粒度のギャップの情報等を得るきっかけとして，要求・実装それぞれが属するプロダクトの組み合わせを照らし合わせることは，重要であると考えられる．そこで，我々は，照合結果を 5 つに分類した．

対象となる派生プロダクト群のプロダクト数をkとして，個々のプロダクトにおける要求の集合をF_iとすると，プロダクト群における全ての要求の集合Fは以下の様に定義できる．

$$F = \bigcup_{i=1}^{k} F_i$$

同様にして，個々のプロダクトにおけるコンポーネントの集合をC_iとすると，プロダクト群における全てのコンポーネントの集合Cは以下の様に定義できる．

$$C = \bigcup_{i=1}^{k} C_i$$

次に，要求の集合Fとコンポーネントの集合Cの対応φを，以下の様に定義する．

$$\varphi : F \to C$$
$$f \in F \to \varphi(f) \in C$$

そして，要求・コンポーネントと，それらの属するプロダクトの組み合わせとの間の関係I_F，I_Cを，以下の様に定義する．Pはk個のプロダクト群の冪集合を表す．

$$I_F : F \to P$$
$$I_C : C \to P$$

要求fの属するプロダクトの組み合わせを$I_F(f)$, 対応するコンポーネント$\varphi(f)$の属するプロダクトの組み合わせを，$I_C(\varphi(f))$とする．これにより，要求・コンポーネント，それぞれが属するプロダクトの組み合わせを照らし合わせた結果は，以下の 5 つに分類される．

Type1.　$I_F(f) = I_C(\varphi(f))$

Type2.　$I_F(f) \supset I_C(\varphi(f))$

Type3.　$I_F(f) \subset I_C(\varphi(f))$

Type4.　$\left(I_F(f) \not\subseteq I_C(\varphi(f))\right) \wedge \left(I_F(f) \not\supseteq I_C(\varphi(f))\right)$

$$\wedge \left(I_F(f) \cap I_C(\varphi(f)) \neq \emptyset\right)$$

Type5.　$\left(I_F(f) \not\subseteq I_C(\varphi(f))\right) \wedge \left(I_F(f) \not\supseteq I_C(\varphi(f))\right)$

$$\wedge \left(I_F(f) \cap I_C(\varphi(f)) = \emptyset\right)$$

出力結果に現れる，それぞれの結果が示唆することとして，考えられることを述べる．組み合わせが一致する Type 1 であれば，その対応関係は確かなものとして，要求・実装間のトレーサビリティリンクとして抽出できる．また，一方が，他方の真部分集合である Type 2 と Type 3 は，要求と実装の粒度のギャップ等の要因を視野に入れた，新たな分析を促す兆候となる．そして，Type 4 と Type 5 の場合は，抽出した対応関係自体が誤りの可能性が高い．これらが大量に現れた場合は，要求を特徴付けるキーワードの再設定や，対象とする構成管理ログの絞込み等を検討する必要が出てくる．

4. 適用実験

4.1. 実験概要

本研究では，まず手法を確立することを目的に，一定規模の派生プロダクト群を対象とした適用実験を行った．したがって，提案手法の対象である大規模な派生プロダクト群に対しても，提案手法が問題なく適用できるかどうかの検証は、今後の課題となる．

実験対象として，オープンソースソフトウェアであり，C 言語の単体テストを支援するテスティングフレームワーク，CUnit[5]を選択した．CUnit のバージョンの内，ver2.0-1，ver2.1-0，ver2.1-2 の 3 つを対象として，要求・実装間のトレーサビリティリンク抽出を行った．実験対象の概要として，それぞれのバージョンにおける，関数数，SLOC，リリース時期を，表 1 に示す．実験には Java で実装したシステムを用いた．コードクローン分析には CCFinderX[6]を利用した．

4.2. 実験結果

4.2.1. 要求資産の共通性・可変性分析

それぞれのバージョン毎に，ドキュメントから要求群を手動で抽出し, 要求のリストを作成した. そして，それらのリストを入力として，共通性・可変性分析を行った．処理時間は 1 秒以下の極短時間であった．

抽出された要求のリストを表 2 に示す．大部分の要求が全てのバージョンに共通したものであったが，表中で*マーク付けられている 4 つの要求は，最新の ver2.1-2 で追加されていたものである．

4.2.2. ソースコードの共通性・可変性分析

それぞれのバージョンのソースコードから，関数を抽出し，関数レベルの共通性・可変性分析，コンポーネントレベルの共通性・可変性分析を行った．なお，類似度の閾値は 0.2 と設定した．まず，関数レベルの結果を表 3 に示す．関数は数が膨大なので，それぞれのプロダクトの組み合わせ毎の関数数を示した. 次に，コンポーネントレベルの結果を表 4 に示す．それぞれのコンポーネントが，どのバージョンに存在しているかを示している．処理時間は，5 回計測した結果，平均で 1 分 37 秒であった．

関数レベルの共通性・可変性分析では，同名の関数であっても，内容の変更により類似度が閾値以下になったことで，共通部ではなくなったケースが多く見られた．具体的には，ver2.0-1 と ver2.1-0，それぞれに固有な、計 11 個の関数については，同名の関数が他バージョンに存在した．しかし一方で，ver2.1-2 に固有な関数には，その他のバージョンに同名の関数がない，完全に固有な関数が多く存在していた．

コンポーネントレベルの共通性・可変性分析では，ほとんどのコンポーネントが共通部として出力された．唯一，Console/Console.c のみが，3 つ全てのバージョンの組み合わせでは，類似度が閾値以下となった．

4.2.3. トレーサビリティリンク抽出

要求・実装それぞれの共通性・可変性分析結果と，構成管理ログを利用して，要求・実装間のトレーサビリティリンク抽出を行った．なお，要求を特徴付けるキーワードの設定は，要求を抽出したドキュメントを利用して，自動的に行った．キーワードは要求 1 つに対して 5 つ設定した．また，構成管理ログのリビジョン数は 156 であり，あまり情報量は多くないので，キ

表 1 実験対象の概要

	関数数	SLOC	リリース時期
2.0-1	221	5931	September, 2004
2.1-0	229	6225	March, 2006
2.1-2	256	7760	October, 2010

表 2 要求リスト

①Activation of suites and tests *
②Adding suites to the the registry
③Adding tests to suites
④Behavior upon framework errors
⑤Cleanup the test registry
⑥Error handling
⑦Getting test results
⑧Initialization the test registry
⑨Lookup of individual suites and tests *
⑩Modifying general runtime behavior *
⑪Modifying other attributes of suites and tests *
⑫Running tests in automated mode
⑬Running tests in basic mode
⑭Running tests in interactive console mode
⑮Running tests in interactive curses mode

表 3 関数の共通性・可変性分析結果

	関数数
[2.0-1]	10
[2.1-0]	1
[2.1-2]	67
[2.0-1, 2.1-0]	39
[2.1-0, 2.1-2]	17
[2.0-1, 2.1-2]	0
[2.0-1, 2.1-0, 2.1-2]	172

表 4 コンポーネントの共通性・可変性分析結果

	2.0-1	2.1-0	2.1-2
/Automated/Automated.c	●	●	●
/Basic/Basic.c	●	●	●
/Console/Console.c [2.1-2]			●
/Console/Console.c	●	●	
/Curses/Curses.c	●	●	●
/Framework/CUError.c	●	●	●
/Framework/MyMem.c	●	●	●
/Framework/TestDB.c	●	●	●
/Framework/TestRun.c	●	●	●
/Framework/Util.c	●	●	●

ーワードが 1 つでも現れたリビジョンのコンポーネントと要求を対応付けた．抽出は，ドメイン毎に行ったが，処理時間はどれも 1 秒以下の極短時間であった．

抽出された要求・実装間の対応関係を表 5 に示す．要求は表 2 で記した番号で表している.表中の記号は，要求・実装の属するプロダクトの組み合わせの照らし合わせの結果を反映しており，○は Type 1，△は Type 2，▽は Type 3 を示している．なお，今回の実験では，Type 4 と Type 5 は存在しなかった．また，記号の塗り潰しの有無は,抽出元リビジョンの分類を表しており，塗り潰されている方は，Type A のリビジョンから抽出した対応であり，塗り潰されていない方は，Type B のリビジョンから抽出した対応である．そして，表中で枠内の色が濃くなっているものは，ドキュメントで明示された,正解となるトレーサビリティリンクである．

4.3. 評価

4.3.1. トレーサビリティリンクの自動抽出

本研究の研究課題 RQ1 が，提案手法により解決されたかの評価を行う．

ドキュメントで明示されていたトレーサビリティリンク 20 個の内，7 割に当たる 14 個を提案手法によって自動的に抽出する事ができた．しかし一方で，明示されていなかった対応関係が多く抽出された．これらが無自覚に形成されたリンクなのか，誤検出であるかの判断は,開発者でないと難しく,今後の課題となる．

Type A のリビジョンから抽出された対応の割合が，あまり多くなかった．このような結果になった原因としては，Type A のリビジョンの絶対数がとても少なかったことが上げられる．156 個のリビジョンの内，それらのリビジョンは 21 個のみであった．したがって，この手法は構成管理ログの性質に大きく左右されると考えられる．

次に，要求・実装が属するプロダクトの組み合わせの照らし合わせの結果の利用価値を考える．今回の実験結果の中で，Type 3 が多く現れているのが，ver2.1-2 で新たに追加された 4 つの要求と，Framework ドメインの対応の部分である.Type 3 の示唆することとして，要求・実装の粒度のギャップが挙げられる．今回のケースでは，新しく追加された要求が，新たな関数の追加によって実現された可能性がある．関数レベルのソースコードの共通性・可変性分析結果を確認すると，確かに Framework ドメインに属する関数が多く追加されていることが分かった．このことから，4 つの新たな要求を，コンポーネントではなく，これらの新たに追加された関数群と結びつけることで，更に精査されたトレーサビリティリンクを抽出することができる．

構成管理ログの利用により，要求・実装間のトレーサビリティリンクについて，ある程度の当たりをつけることが可能であると確認できた．しかし，Type B のリビジョンから得た情報についての対応や，要求・関数間のトレーサビリティリンク抽出には，ユーザーへの負担が依然として残されているので，対応を検討する必要がある．

4.3.2. 実行時間

本研究の研究課題 RQ2 が，提案手法により解決されたかの評価を行う．

最も実行コストがかかったのが，ソースコードの共

表 5 要求・実装間の対応関係

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮
/Automated/Automated.c		○	○			○			▽			●			
/Basic/Basic.c		○	○	●					▽				●		
/Console/Console.c [ver2.1-2]						△						△			△
/Console/Console.c						△						△			△
/Curses/Curses.c						○					▽	○			○
/Framework/CUError.c															
/Framework/MyMem.c	▼	●	●	●	●			●	▼		▽	○			
/Framework/TestDB.c	▽	●	●	○	○	●			▼	▽	▽	○	○	○	○
/Framework/TestRun.c	▽	●	●	○	○	○			▼	▽	▽	○	○	○	○
/Framework/Util.c															

通性・可変性分析であったが，1 分 37 秒という実用的時間内に収まった．要求資産の共通性・可変性分析と要求・実装間のトレーサビリティリンク抽出にかかるコストは，今回の実験対象レベルの規模では，限りなく小さいことが分かった．

以上より,数千行規模の派生プロダクト群に対して，実用的時間内にトレーサビリティリンクを抽出できたといえる．しかし，派生プロダクト群の規模が，数十万，数百万規模と大きくなった場合，処理時間の増加は線形的ではないと考えられる．今回処理時間の大部分を占めた,ソースコードの共通性･可変性分析では，コードクローンの数が，処理時間に大きな影響を与える．規模が大きくなれば，関数数は増え，組み合わせの増加によってクローン数は指数関数的に増えていく．したがって，対象となるプロダクト群の大規模化に対しては，ソースコードの共通性・可変性分析をドメイン毎に分けて行う等の，追加対策が必要となり得る．

5. 関連研究

確率モデルやベクトル空間モデル等の情報検索技術を用いて，ソースコードとドキュメントのトレーサビリティリンクの復元を行う研究がある[7]．ソースコード中の識別子と，ドキュメント中の言葉を比較することで，ソースコードとドキュメントを対応付けている．一方，我々は，要求資産とソースコードの対応付けを，構成管理ログを仲介して行っている．

海谷らは，情報検索技術を用いて，要求変更によるソースコードへのインパクトを分析する手法を提案している[8]．彼らは，日本語で書かれた設計書を仲介して，要求とソースコードの対応関係を抽出している．よって，設計書に識別子や関数名等が表れていれば，要求を関数と直接対応付ける事ができる．一方，我々は，仲介物として構成管理ログを用いている．構成管理ログでは，要求レベルの概念を表す語と，実装レベルの概念を表す語が，メッセージとパスという形で，明確に区別されているので，適用対象の言語は日本語に限定されない．

6. おわりに

我々は，大規模な派生プロダクト群における，要求・実装間のトレーサビリティリンクを自動的に抽出する手法を提案した．提案手法では，構成管理ログを対象に情報検索を行うことで，要求・実装それぞれの共通性可変性分析結果の間の対応関係を抽出する．そして，抽出された対応関係に付加された情報を元に，要求・実装間のトレーサビリティリンクを抽出する．提案手法により，新たな派生プロダクトの開発を支援することができる．また，一定規模の派生プロダクト群を対象として，適用実験を行った結果，実用的な時間内で,妥当なトレーサビリティリンクを抽出できた．

今後の課題として，本研究の対象である，大規模な派生プロダクト群を対象に，適用実験を行い，提案手法の有効性を確認する必要がある．また，現状では，要求と対応する関数の特定は，関数の共通性・可変性分析結果を元に人手で行っているので，自動的に候補となる関数群が抽出される形を目指す．

参考文献

[1] K. Kumaki, R. Tsuchiya, H. Washizaki and Y. Fukazawa, “Supporting Commonality and Variability Analysis of Requirements and Structural Models,” MAPLE 2012, SPLC’12, vol.2, pp.115-118, 2012.

[2] G. Salton and M. J. McGill, “Introduction to Modern Information Retrieval,” McGraw-Hill, New York, 1983.

[3] 山本哲男，松下誠，神谷年洋，井上克郎, “ソフトウェアシステムの類似度とその計測ツール SMMT,” 電子情報通信学会 D-I, no.6, pp.503-511, 2002.

[4] 吉村健太郎, Ganesan, D. and Muthig, D, “プロダクトライン導入に向けたレガシーソフトウェアの共通性・可変性分析法,” 情報処理学会論文誌, vol.48, no.8, pp.2482-2491, 2007.

[5] CUnit，http://sourceforge.net/projects/cunit/

[6] CCFinderX，http://www.ccfinder.net/

[7] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia and E. Merlo, “Recovering Traceability Links between Code and Documentation,” IEEE Transactions on Software Engineering, vol.28, no.10, pp.970-983, 2002.

[8] 海谷治彦，長田晃，原賢一郎，海尻賢二, “要求変更によるソースコードへのインパクトを分析するシステムの開発と評価,” 電子情報通信学会 D, vol.J93-D, no.10, pp.1822-1835, 2010.