情報処理学会論文誌 Vol.53 No.2 653–661 (Feb. 2012)

ソフトウェアが中心でない製品における既存技術を利用したソフトウェア改訂支援

海谷 治彦[1,a)] 原 賢一郎[1] 小林 亮太郎[1] 長田 晃[1] 海尻 賢二[1]

受付日 2011年5月25日, 採録日 2011年9月12日

概要：ソフトウェアを含め，ほとんどの工業製品は以前の製品を改訂し開発されている．よって，既存ソフトウェアの改訂を支援することも重要である．革新的なソフトウェア改訂技法や枠組みが提案はされているが，それらを現実の開発に導入することは現状の開発体制の観点から容易でない場合が多い．特にソフトウェアが中心でない製品の場合，その傾向は顕著である．本稿では，ソフトウェアが中心でない製品におけるソフトウェア改訂において，どのように技術導入をすべきかについての調査および試行結果を報告する．初めに我々はソフトウェアが中心でない製品を開発する産業界の協力者の支援をうけ，どのような作業をどのような技術で支援すべきかを調査した．結果，インパクト分析作業が技術的に支援可能であり，情報検索技術（IR）を用いたトレーサビリティ技術で支援することが適切であると判断した．次に協力者から提供があった実開発のデータに対して，当該技術を適用し，協力者とともに問題点や改善点を模索した．結果として，要求変更の特徴づけを支援するための技術文書の索引付け，IR の入力データの改善を支援する機械学習，間接的なインパクトを知るためのソースコード上の静的解析の 3 つが，IR に基づくトレーサビリティを改善する追加技術として適切であると判断した．我々はこれら 3 つの技術を追加したインパクト分析支援ツールを試作し，産業界の協力者に評価を依頼し，期待どおりの改善が見込まれることを確認した．

キーワード：変更波及分析，情報検索，索引化，トレーサビリティ，機械学習

Supporting Software Revision in Software Non-intensive Projects Using Existing Techniques

Haruhiko Kaiya[1,a)] Kenichiro Hara[1] Kyotaro Kobayashi[1] Akira Osada[1]
Kenji Kaijiri[1]

Received: May 25, 2011, Accepted: September 12, 2011

Abstract: Most industrial products are developed based on their former products including software. Revising existing software according to new requirements is thus an important issue. However, innovative techniques for software revision cannot be easily introduced to projects where software is not a central part. In this paper, we report how to explore and apply software engineering techniques to such non-ideal projects to encourage technology transfer to industry. We first show our experiences with industrial partners to explore which tasks could be supported in such projects and which techniques could be applied to such tasks. As a result, we found change impact analysis could be technically supported, and traceability techniques using information retrieval seemed to be suitable for it. We second had preliminary experiences of a method using such techniques with data in industry and evaluated them with our industrial partners. Based on the evaluation, we third improved such a method by using following techniques; indexing of technical documents for characterizing requirements changes, machine learning on source codes for validating predicted traceability and static source code analysis for finding indirect impacts. Our industrial partners finally evaluated the improved method, and they confirmed the improved method worked better than ever.

Keywords: change impact analysis, information retrieval, indexing, traceability, machine learning

1. はじめに

自動車や家電製品にソフトウェアが組み込まれていることは現在では当たり前のこととなったため，これらの製品におけるソフトウェア開発の改善は重要な要素の 1 つとなっている．また，これらの製品開発費に占めるソフトウェアの割合も増加している．しかし，実際の，これらの製品開発において，ソフトウェアは必ずしも中心的であるとはいえないことを我々は実際の開発の調査 [1] を通して確認してきた．どのような製品のソフトウェアでも今日では以前の製品で利用されたソフトウェアを改訂することで開発されることがほとんどである．モデル駆動開発（MDD）やソフトウェアプロダクトライン技術（SPL）のような，ソフトウェアの改訂を効率的かつ系統的に行う革新的な技術は数多く提案され，一部は産業界にも適用されている．しかし，ソフトウェアが中心でない製品においては，その導入は容易でない．その理由の 1 つは，革新的手法の多くは，既存の開発プロセスや手法を大きく変更することを要求しており，それらの変更を実際には受容できない場合が多いからである．ソフトウェアが中心でない製品開発では，電気電子部品，機械部品等の他の部品開発との依存性のため，開発プロセスの変更はさらに困難なものになる．このような背景に鑑みて，従来のソフトウェア改訂作業を大きく変更せずに，その作業を改善するための手法とツールの開発を産業界の協力者とともに我々は進めてきた．本稿では，最初に提案した支援法およびツール [1] の問題点を明らかにし，既存の技術を利用することで，その改善を行った結果を報告する．

本稿の構成は以下のとおりである．次章で，我々が過去に行ったソフトウェアが中心でない製品における開発現状の調査と，関連する研究について言及する．3 章では我々が最初に開発および評価を行ったインパクト分析支援法とツール [1] の概要とその評価を概説する．その評価をふまえ，手法の改善要求を明確化する．4 章では改善要求を達成する解法の提案を通して，インパクト分析支援法第 2 版を提案し，5 章でその遂行支援ツールを紹介する．6 章で支援法第 2 版とツールの評価を述べ，最後にまとめと今後の展望を述べる．

2. インタビューおよび調査

ソフトウェアが中心でない製品におけるソフトウェアの改訂を支援する方法を模索するために，我々はいくつかの組み込み製品会社の技術者にインタビュおよびアンケート調査を行い，現状や支援すべき点を明らかにした．結果の詳細は文献 [1] で発表済みであるが，次章，および 4 章での支援法および，その改善方向を明確にするために，本章で，その概要を紹介する．

最初に，改訂のための環境は理想的ではないことが確認できた．たとえば，文書やソースコード等は，構造の悪さから，改訂に適していない場合が多いことが分かった．しかし，既存の製品の動作実績があることから，文書やソース構造の大胆な整理を行うことが許されない場合が多い．また，開発計画が短いため，構造の整理を行うことに制限がないとしても，それを行う時間や工数の捻出が困難であることが分かった．

第 2 に，改訂のための文書やソース等のリソースについて概観する．要求や設計の文章は自然言語で記述されており，改訂中にも参照可能である．しかし，特に要求変更については，電子メールや口頭等の非形式的な様式で扱われる場合も多いことが分かった．擬似コードや図表，式，UML 図等の構造的な表現が要求仕様書や設計書に含まれることもあるが，構造的な表現はあくまで自然言語文書の補足情報として扱われる．調査対象の製品は組み込み製品であるため，コードは C もしくはその拡張言語で書かれており，管理単位は関数となる．上記に述べるような文書やコードの特徴から，文書やコード間のトレーサビリティは完全には管理されていない．

第 3 にソフトウェアの更新方法の現状について概観する．変更要求に基づき，修正すべき箇所を発見し，修正するという手順を基本的には踏んでいる．文書やコード間のトレーサビリティが完全には管理されていないため，修正箇所の発見は容易ではない．また，既存の文書やソースの構造の悪さが直せない場合が多いため，修正自体も困難な場合が多いうえに，非技術的な制約（動作実績があるため大幅な構造変更が困難であること）がある．修正箇所の発見は，レビュー等を通して手動でトレーサビリティを追跡し，修正箇所を発見している．すなわち，要求変更によるインパクト分析のためのレビューを行っている．設計書は要求変更とインパクトを受けるコードとの仲介のために利用される．少数の事例では改訂対象のソフトウェアの動的解析がツールによって行われているが，そのほかでは，ツールはほとんど利用されていない．改訂対象のソフトウェア開発にかかわった技術者と連絡をとることは，多くの事例で可能であるようである．

上記の調査結果に基づき，ソフトウェア改訂支援において，自然言語による文書（設計書等）とソースコード上におけるインパクト分析が技術的に支援できる問題であると我々は認識した．また，変更要求，文書，ソースコード間のトレーサビリティがインパクト分析を行うに際して有用であると分かった．また，現状の作業方法や工数の余裕，非技術的な制約を考えると，MDD や SPL 等の形式的もしくは系統的なソフトウェアの改訂支援の導入は，現時点では時期尚早と判断した．

[1] 信州大学
Shinshu University, Nagano 380–8553, Japan
[a)] kaiya@shinshu-u.ac.jp

そこで，我々はトレーサビリティに関する既存技術の調査を行った．ソフトウェア保守の研究分野では，形式的概念分析（formal concept analysis）[2] やモデルに基づくアプローチ [3], [4] を用いたトレーサビリティ技術がよく知られている．しかし，これらの技術の多くは対象となる文書やソースコードに対しての事前準備を少なからず必要とする．トレーサビリティリンクを半自動的に生成する技術も進められており，それらは事前準備の低減に貢献する．トレーサビリティリンクを半自動的に生成する技術は大まかに，静的技術，動的技術，複合的な技術の 3 種類に分類できる．上記の調査結果に鑑みると動的および複合技術をつねに用いることはできない．よって，静的技術が我々の支援対象に適切である．情報検索（IR）を用いた技術 [5], [6], [7], [8] は静的技術の典型的なものであある．たとえば研究 [6], [7] では確率モデルが用いられており，研究 [6] ではベクトルモデル，研究 [8] では Latent Semantic Indexing（LSI）が用いられている．これらの研究での技術は卓越したものではあるが，少なからず事前準備が必要となり [9]，それが産業界への浸透が進まない大きな原因となっている．トレーサビリティに関係するツールの試作も行われている．たとえば要求分析工程におけるツール [10] や，コーディング段階でのツール [11] 等である．しかし，異なる工程の成果物間のトレーサビリティに関するツールは少ない．また，これらのツールの多く [12] は成果物が UML 等の構造的な様式で記述されていることを想定している．我々の調査にあるように構造的な様式はあくまで補足情報であるため，我々が対象としている支援対象の開発には既存のツールを適用することは困難であると思われる．

3. IR を用いたインパクト分析支援の試行と改善点

前章で述べたインタビュおよび調査の結果から，ソフトウェアが中心でない製品におけるソフトウェアの改訂において，情報検索（IR）に基づくトレーサビリティ技術を用いてインパクト分析を支援することが重要であると我々は判断した．そこで，我々は IR を用いたインパクト分析支援手法を考案し，その支援ツールを開発し，開発が終了した実プロジェクトデータを分析することで手法とツールの評価を行った [1]．その評価結果をもとに，インパクト分析支援法をどのように改善すべきかに関する要求獲得を，産業界の協力者とともに行った結果を本章で紹介する．

要求獲得の前提を明確にするために，既発表 [1] であるインパクト分析支援法とその評価について概説する．**図 1** に示す例を用いて我々のインパクト分析法を説明する．前章で述べたように要求変更は自然言語記述や口頭で与えられる．そこで分析の入力は図に示すように自然言語の文もしくは文章であるとし，それらの文や文章を語句の集合として特徴付ける．図では顧客やハードウェア等の他の部品

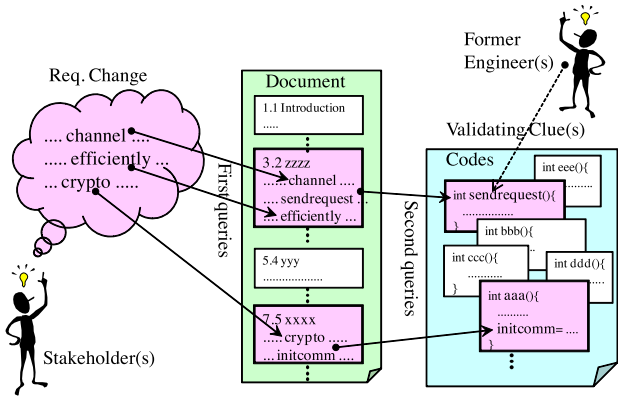

図 1 設計書を仲介としたインパクト分析（設計書は節の集合と見なしコードは関数の集合と見なす）

Fig. 1 Impact analysis using design documents as a mediator (Document is a set of sections and codes are set of functions).

にかかわる技術者等のステークホルダが，要求変更を特徴付ける文を提示し，それが 3 つの語句として特徴付けられている．これは前章でも述べたように，現状の開発作業でも同様のことがソフトウェア技術者によって行われている．要求変更に関する語句と実装の語句との間にはギャップがあるが，ソフトウェア技術者は，設計書を読むことで，そのようなギャップを埋める．そこで，我々の手法でも設計書を要求と実装のギャップを埋めるために用いる．具体的には図 1 に示すように，設計書を節の集合と見なし，それぞれの節における語句の共起関係に基づきギャップを埋める．図の例においては，“channel”，“efficiently”，“crypto” の 3 つの語句が要求変更を特徴付けているが，これらは，設計書の 3.2 節，7.5 節に出現する．そこで，これらの節に出現するソースコードに関連する識別子（関数名や変数名）をキーワードとして関数群を検索し，変更が及ぶと思われる関数（impacted-prone functions）を予測する．我々の手法では設計書の検索に用いられる要求変更を表す語句を First query，関数群の検索に用いられる語句を Second query と呼んでいる．実際の開発では，開発前任者や関係者と情報交換をすることで，明らかにインパクトを受ける関数群の一部を手がかり（clue）として事前に知ることができる．我々の手法では，このような手がかりを validating clue と呼び，検索結果の妥当性を確認するのに用いる．

上記のインパクト分析手法の詳細は文献 [1] に発表済みであるが，ここでは結果の概略を紹介する．first query はソフトウェア技術者が要求変更文に基づき，選択していかなければならない．この first query の数を増やせば，当然，実際にインパクトがある部分を漏れなく検索できる（recall が 100%に近づく）が，検索誤りも増加してしまう（precision が低くなる）．本手法は，ソフトウェア改訂におけるソースコードレビューの範囲の絞り込みや，レビュー工数の優先的な分配（より疑わしい場所を時間をかけてレ

ビューする）の支援を目的としている．そこで，recall は100%に近い状態でありつつ，関数全体として比較的少ない割合（我々はこの値を coverage と呼ぶ）がインパクトありと予測されるのが望ましい．現実の開発におけるデータに本手法を用いて適用したところ，recall が 100%において，precision が 30%程度，coverage が 20%程度となった．この結果を当該データの提供者に吟味してもらったところ，妥当な数値であるという評価を得た．coverage が 20%程度ということは，全ソースの 2 割を非常に重視してレビューすればよいという指針となる．実際の開発では真にインパクトを受ける部分を事前に知ることはできない．よって，recall，precision を知ることはできない．これでは，first query をどの程度にすべきかの指針が得られない．そこで，前述の validating clue を正解と見なし，擬似的な recall と precision（本稿では reall' precision' と表記する）を求め，それを指針として，first query の数と種類を適切に設定できるかを評価した．結果として，recall' と precision' は first query を適切に設定する指針として有効であることを確認した．

文献 [1] を発表後，我々は産業界の協力者とともに手法と評価結果について，さらに検討を行った．協力者は手法と結果についておおむね満足であった．また，複雑な作業や付加的な事前準備がほぼ必要ないため，現実の開発への適用可能性についても肯定的であった．加えて本手法を改善するため以下にあるような要求を提示された．

- 改善要求 R1：first query の選択支援が欲しい．図 1 に示すように，要求変更を特徴付けるため，いくつかの語句を選択しなければならない．しかし，要求変更の文書や口頭の伝言内容が，設計書における語句を利用して述べられているとは限らない．よって，設計書の検索に適した語句を，要求変更を特徴付けるために選択するのは容易ではないという意見を得た．
- 改善要求 R2：validating clue を改善する仕組みが欲しい．実際にインパクトを受ける関数を事前に知ることはできないため，validating clue の役割は大きい．また，validating clue がより多く，正確であるほど，その効果は大きい．しかし，前任者からより多く正確に validating clue を得るためのヒントや手法がないため，validating clue の改善が実際問題として難しいという意見があった．
- 改善要求 R3：間接的なインパクトを予測する機能が欲しい．この時点では要求変更に直接影響を受ける部分のみを注目しているが，コストや工数の観点からは間接的な影響も知る必要がある．

4. インパクト分析法第 2 版

前章での改善要求 R1，R2，R3 に基づき，我々は図 1 の分析手法を以下の観点から改善した．

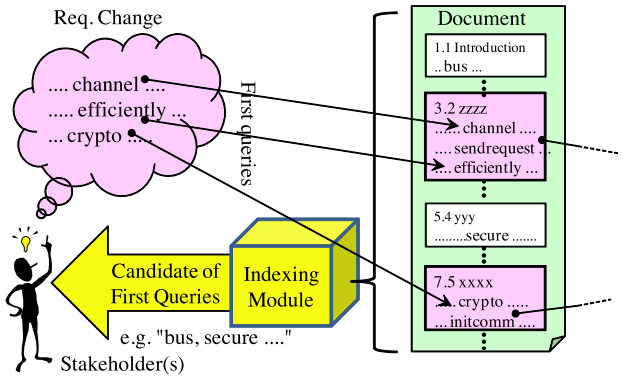

図 2 解決案 S1：first query の選択を支援するための索引付け
Fig. 2 Solution S1: indexes of documents for choosing first queries.

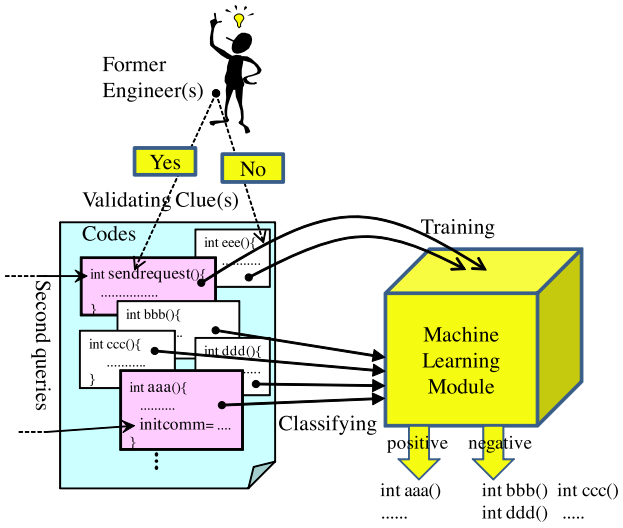

図 3 解決案 S2：clue 改善を支援する機械学習
Fig. 3 Solution S2: machine learning for expanding information of clues.

- 解決案 S1：ソフトウェアを改訂する技術者が first query を選ぶ支援を行うため，設計書の索引付けを行い，その索引を first query の候補として技術者に提示する機能を追加する．**図 2** の例では，設計書中の “bus” や “secure...” といった語句を索引として選び，それを要求変更を特徴付ける語句である first query の候補として技術者に提示している．first query は要求変更を特徴付ける語句であるため，関数名や変数名等の実装に近い語句が設計書に含まれていても，first query の候補からは除外する．
- 解決案 S2：前任者等から得られる validating clue を改善するため，機械学習のモジュールを導入し，validating clue の候補を，改訂を行う技術者が前任者等に積極的に照会する支援を行うこととした．**図 3** に概要を示す．前任者から，明らかに変更が必要な関数（図中の “Yes” に相当）だけでなく，明らかに不要な関数（図中の “No” に相当）の情報も可能なら獲得する．これらの情報をもとに，機械学習に基づき，Yes

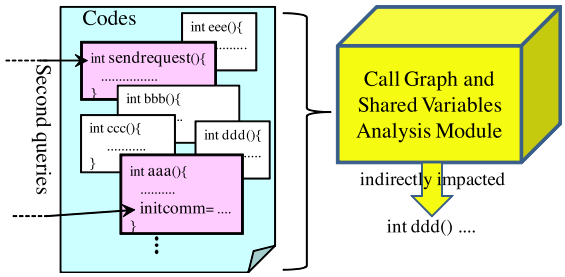

図 4 解決案 S3：間接的インパクトの発見のための静的コード解析
Fig. 4 Solution S3: static source code analysis for finding indirect impacts.

かもしれない関数の候補（図中の “positive”）と，No かもしれない関数の候補（図中の “negative”）を予測する．この予測の是非を再度前任者に照会することができれば，validating clue の量と精度を段階的に改善することができる．機械学習を行うためには，個々の関数の特徴付けを行わなければならないが，我々は関数中の出現語句とその頻度によって関数を特徴付けることとした．

- 解決案 S3：間接的なインパクトを知るために，**図 4** に示すように，ソースコード中の静的な情報（呼び出し関係と共有変数）を用いることとした．これは，改訂対象のソフトウェアのすべてが動的解析可能とは限らないためである．間接的なインパクトの度合いは文献 [13] にある影響波及解析の考え方を利用することとした．文献 [13] では，ある関数（文献ではモジュール）から他の関数への影響の波及度合いに基づき，影響があると思われる関数によって，他の関数がどれだけ影響を受けるかを計算する．本分析手法においても，間接的なインパクトを知りたい理由が，コストや工数の目安をつけることであるため，文献 [13] の手法は適切である．

5. インパクト分析第 2 版のための支援ツール

前章での改善案を含めたインパクト分析手法第 2 版を実施するための支援ツールを試作した．本章ではツールの利用法および実装について説明する．

5.1 利用法

本ツールはソフトウェアの改訂を行う技術者が図 1 に示すようなインパクト分析を行う際に用いる．本ツールは前章で紹介した 3 つの解決案を実現する機能を有する．**図 5** と**図 6** の典型的な画面例を用いて，以下に典型的な利用法を紹介する．ここでは，説明の便宜のため FTP クライアントソフトウェアに，暗号化通信の機能を追加するソフトウェアの改訂を例として用いている．

(1) 図 1 にあるように最初に技術者は first guery を要求変更を特徴付けるために指定する．この first query は，図 5 の一番上の first query とラベル付けされたウインドウに書き込むことで指定できる．この例では “send receive connect” の 3 つの語句が指定されている．first query は技術者が自由に書き込んでもよいし，続くステップにおいて索引から選択してもよい．

(2) 図 2 に示した索引付けモジュールによって，first query 候補が自動的に設計書から抜き出され技術者に提示される．この候補は図 5 中の FirstQuery Selector とラベル付けされたウインドウに表示される．選択された語句は自動的に前ステップで紹介した first query とラベル付けされたウインドウに転記される．技術者は転記された first query 候補を自由に編集できる．

(3) ここまでのステップにおいて選択された first query に基づき，ソースへのインパクトを仲介する節の候補（図 1 における 3.2 節や 7.5 節）が自動的に選択される．選択された節候補は図 5 中の Section とラベル付けされたウインドウに表示される．このウインドウ中の各行が節の候補に相当し，それぞれの行には，候補を絞るための情報として，出現した query の種類数（第 3 カラムの Query Type）と，出現した query の総数（第 4 カラムの Query Cou...）が示されている．たとえば，2.10 節は，“send receive connect” の 3 種類のうちの 2 種類が出現し，その出現総数は 19 であるということが分かる．第 5 カラムのチェックを外すことで，候補から外すことができる．第 6 カラムのチェックを入れることで，first query を再検討し，節の候補を再検索した際でも，候補としないことも指定できる．

(4) 前ステップで仲介となる節の確定後，ソースコードを検索するための second query の候補が自動的に選ばれる．この候補は図 5 中の Second Query とラベル付けされたウインドウに提示される．second query は設計書の節とソースコード中の関数それぞれに出現し，設計書とソースコードのトレーサビリティを確立するのに用いられる．よって，極端に多くの関数に出現したり，極端に多くの節に出現したりする second query は，その語句が当該ソフトウェア中で，一般的すぎるため，特定のトレーサビリティ確立に役立たない場合がある．本ツールでは，それぞれの second query が，どの程度の節および関数に，どのような分布で出現するかを技術者に提示することで，second query の取捨選択を支援する．

図 5 中の Second Query とラベル付けされたウインドウを例として，この情報について説明する．この画面例ではいくつかの second query の候補が表示されているが，候補 type は 6 個の関数に出現し（第 3 カラム），この 6 個は全関数の 1.449%（第 4 カラム）にあたる．一方，type は 51 個の節に出現し（第 10 カラム），この 51 個は全節の 9.701%（第 11 カラム）にあ

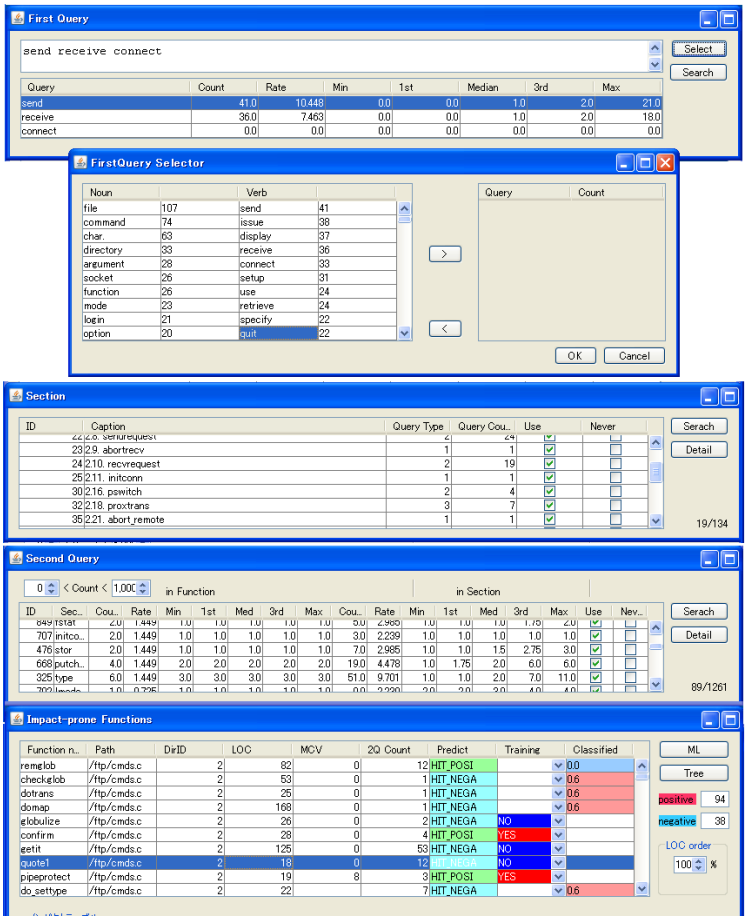

図 5 支援ツールの主な画面例

Fig. 5 A snapshot of a main window of our supporting tool.

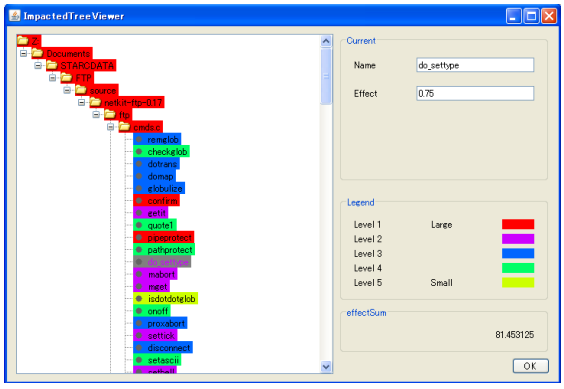

図 6 支援ツールにおける間接的なインパクトの可視化

Fig. 6 A snapshot of a window for visualizing direct and indirect impacts.

たる．この情報から技術者は，type を second query として利用すべきか否か判断するわけだが，語自体が一般的なものである点，および，全節の 10%近くに出現することから，候補から除外したほうがよいかもしれない．なお，Second Query とラベル付けされたウインドウの第 5 から 9 カラムは関数におけるそれぞれの second query の出現候補に関する分布，および，第 12 から 16 カラムは節におけるそれぞれの second query の出現候補に関する分布に関する情報を提示している．また，同ウインドウの左上の範囲指定をすることで，個々の関数に出現する second query 数に基づいて，second query の候補を事前に絞ることができる．図中では，範囲が 0〜1000 であるため，事実上，この機能による絞り込みを行っていない．

(5) 前ステップにおいて second query が確定した後，それに基づいてインパクトがあると思われる関数（impacted-prone functions）が自動的に検索される．図 5 の一番下にある “Impact-prone Functions” とラベル付けされたウインドウがこの検索結果の例である．実際にインパクトがあるか否かの判断に有用な情報が，それぞれの関数候補に付加されている．第 4 から 6 カラムは，それぞれ，行数（LOC），MaCabe Cyclomatic Value（MCV），second query の出現数（2Q Count）が示されている．

図 5 には表示されていないが，別途，validating clue（前任者等から明らかにインパクトがあると告げられた関数）をツールに入力することで，“Impact-prone Functions” とラベル付けされたウインドウの予測結果の妥当性確認を行うことができる．同ウインドウの

第7カラム（Predict）が，HIT_POSIとなっている関数は，与えられたvalidating clueにも含まれるため，インパクトを受ける可能性が妥当である関数である．HIT_NEGAとなっているものは，それ以外である．Validating clueは，実際にインパクトがある関数の一部であると考えられるため，HIT_NEGAだとしても，インパクトがある可能性はある．よって，前任者等にいくつかの候補についてインパクトがあるか否かを確認することで，validating clueの情報を拡充することができる．

“Impact-prone Functions”とラベル付けされたウインドウの最後の2つのカラム（TrainingとClassified）は，機械学習を用いて，validating clueを拡充するための情報を得る様子を示している．この例では，globulizeからpipeprotectまでの5個の関数について，改めて，前任者等にインパクトがある（YES）か否か（NO）の情報を確認し，その情報をトレーニングデータとして，機械学習を行うことで，他の関数がインパクトがあるか否かの予測を行っている．予測結果は最後のカラムに表示されており，0に近い場合は，インパクトの可能性が低い，1に近い場合は，高いことを示している．

(6) 図5の“Impact-prone Functions”とラベル付けされたウインドウの右のTreeとラベル付けされたボタンを押すことで，図6に示すような間接的なインパクトが数値と色で示される．要求変更から直接インパクトをうける関数のインパクトの度合いを1として，それぞれの関数のインパクトの度合いが0から1の間で示される．ある関数から別の関数への影響量の値は，関数呼び出しがある場合と，共有変数がある場合の2通りについて係数を自由に設定できる．図6の例では，関数呼び出しの係数を0.5，共有変数の場合を0.75としている．たとえば，要求変更に直接関係がある関数と共有変数を持つ関数は，0.75のインパクトの度合いを持つと見なす．一方，要求変更に直接関係がある関数から呼び出されている関数から，さらに呼び出される関数は，$1 \times 0.5 \times 0.5 = 0.25$のインパクトの度合いを持つと見なす．

5.2 実装について

このツールを実装では，JavaおよびJavaのGUIコンポーネントを利用した．図2の索引付けのためには，文書を形態素解析する機能を含むKHコーダ[14]を改造したものを利用した．図3の機械学習のためには，WEKA[15]を利用した．ソースコードの静的解析は，doxgen[16]の解析結果を利用し独自に実装した．ツールのデータ管理にはポータビリティを考慮して，SQLite[17]を利用した．SQLiteはサーバを動作させる必要がなく，通常のファイル操作と同様にデータベースを扱うことが可能となる．

6. インパクト分析法第2版の評価

4章および前章で紹介したインパクト分析第2版およびその支援ツールを評価するために，3章で紹介した最初の分析法と分析法第2版の双方を産業界の協力者に利用してもらうことで，その比較を行った．適用したデータはすでに開発が終了した組み込みシステム製品であるため，実際にインパクトがあった関数（正解）が明らかになっている．実際にインパクトがあった部分は新旧ソースコードの差分をとり識別した．よって，実際にインパクトがあった関数（正解）は要求変更から直接影響を受けた部分だけではなく，間接的に影響を受けた部分も含まれる．

分析法第2版およびその支援ツールのほうが，3章の分析法およびツールよりも多くの機能を有している．よって，第2版の分析結果のほうが優れていることは予想できる．一方，第2版のほうが機能等が多いため，利用する技術者の負担も増加すると思われる．そこで，適用のしやすさや使いやすさ等の協力者の主観的な評価が重要となる．

まず，**表1**にrecallおよびprecisionを用いたインパクト予測品質の比較を示す．最初の分析法の結果は，first queryの選択によってprecisonとreallがどのように変動するかの調査時に測定したものであるため，実務者から複数の結果が示されていた．このような事情から，実務者自身がベストと判断した1つの結果はない．そこで，本稿では，recallが第2版の結果と同程度のものを比較対象とした．最初の分析法の結果の中には，第2版よりrecallが良い結果もあったが，そのような結果はprecisionが悪いため比較対象とはしなかった．一方，第2版の結果は実務者がベストと思われる1つの結果を提出してもらった．なお，機械学習の結果は本来の予測と独立しているため，表1に別途結果を示している．むろん，すべての実験中において，recallとprecisionは実務者は知ることはできない．これらの値は実験後に実験者が計算した．

同程度のrecall値の最初の分析法の結果と比較した場合，第2版はより良いprecisionを示している．これより，軽微ではあるが，結果は改善されたと考えられる．また，分析法第2版の通常のrecall，precisonの結果と，機械学習（machine learning）を用いた結果を比較すると，recallに変化はないが，precisionは微増している．よって，機械学習は検索精度の向上に貢献していると考えられる．

上記の比較評価の後，インタビュを通して，我々は手法の適用を行った協力者に適用のしやすさや使いやすさの観点から，2つの手法の主観的な評価を得た．図2に示した要求変更を特徴付けるための語句（first query）選択支援について，特に大きな評価を得た．最初の分析法の特性上，要求変更を顧客や関係技術者の文言のまま特徴付けても，うまく検索ができないことが多かった．よって，first queryの選択作業は大きな負担であったようである．しか

表 1 最初の分析法と分析法第 2 版のインパクト予測品質の比較
Table 1 Quality of the results of a preliminary method in Figure 1 and improved method in Figures 2, 3 and 4.

(%)	recall	precision	recall about machine learning	precision about machine learning
3 章の最初の分析法（recall が 2 版と同程度）	87	8	-	-
分析法第 2 版	87	11	87	13

し，第 2 版で導入された設計書の出現単語に基づく索引によって，協力者自身が同義語や言い回しの違いを考慮しつつ，適切に語句を選択できたようである．ツール上の操作は増加したが，作業負担の軽減になったようである．また，関数を選択するための second query の取捨選択については，second query の出現する節や関数の数や，その統計情報がおおいに役立ったという評価を得た．

7. 結論

本稿では文献 [1] で発表したソフトウェアのインパクト分析支援法を改善した分析法第 2 版の考案と評価を報告した．本稿で扱う分析法はソフトウェアが中心ではない製品の開発において，従来のソフトウェア開発の流れを大きく変更せずに，ソフトウェア改訂作業を改善しなければならないという制約のもとで効果的である．本稿の第 1 の貢献点は，現場の技術者による試験と評価が，既存手法の改善に貢献した報告している点にある．これによって，現場の技術者による既存技術の試験と評価が促進されることを期待している．第 2 の貢献点は，索引付け，機械学習，静的解析等，既存技術の追加でさえ開発支援手法の改善となる実例を示したことである．これによって，新規性だけでなく有用性や適用法に着目したソフトウェア開発支援に関する研究が促進されることを期待する．

本稿で提案したインパクト分析支援法では，要求変更とソースコード間のトレーサビリティをとるため，設計書を仲介として利用している．設計書の役割は，むろんこのようなトレーサビリティをとることだけではない．しかし，今後の課題として，トレーサビリティのとりやすさの観点から，設計書の品質を測定する必要がある．実際，産業界の協力者からは，ソフトウェア改訂を効率的に行うために，どのような様式の設計書を書くべきかの指針がないことが問題点として指摘されていた．トレーサビリティのとりやすさの観点から，設計書の品質を測定することは指針の 1 つとなりうる．

謝辞 本研究は半導体理工学センター（STARC）の支援のもとで行われた．

参考文献

[1] 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，電子情報通信学会論文誌，Vol.J93-D, No.10, pp.1822–1835 (2010).
[2] Niu, N. and Easterbrook, S.M.: Concept analysis for product line requirements, *AOSD*, pp.137–148 (2009).
[3] Metzger, A., Heymans, P., Pohl, K., Schobbens, P.-Y. and Saval, G.: Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis, *RE*, pp.243–253 (2007).
[4] von Knethen, A.: Change-Oriented Requirements Traceability: Support for Evolution of Embedded Systems, *ICSM*, pp.482–485 (2002).
[5] Abadi, A., Nisenson, M. and Simionovici, Y.: A Traceability Technique for Specifications, *ICPC*, pp.103–112 (2008).
[6] Antoniol, G., Canfora, G., Casazza, G., Lucia, A.D. and Merlo, E.: Recovering Traceability Links between Code and Documentation, *IEEE Trans. Softw. Eng.*, Vol.28, No.10, pp.970–983 (2002).
[7] Tang, A., Jin, Y., Han, J. and Nicholson, A.E.: Predicting Change Impact in Architecture Design with Bayesian Belief Networks, *WICSA*, pp.67–76 (2005).
[8] Marcus, A. and Maletic, J.I.: Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing, *ICSE*, pp.125–137 (2003).
[9] Rovegard, P., Angelis, L. and Wohlin, C.: An Empirical Study on Views of Importance of Change Impact Analysis Issues, *IEEE Trans. Softw. Eng.*, Vol.34, No.4, pp.516–530 (2008).
[10] Tanabe, D., Uno, K., Akemine, K., Yoshikawa, T., Kaiya, H. and Saeki, M.: Supporting Requirements Change Management in Goal Oriented Analysis, *RE*, pp.3–12 (2008).
[11] JRipples, available from ⟨http://jripples.sourceforge.net/⟩ (accessed 2010-12).
[12] von Knethen, A. and Grund, M.: QuaTrace: A Tool Environment for (Semi-) Automatic Impact Analysis Based on Traces, *ICSM*, pp.246–255 (2003).
[13] 早瀬康裕，松下 誠，楠本真二，井上克郎，小林健一，吉野利明：影響波及解析を利用した保守作業の労力見積りに用いるメトリックスの提案，電子情報通信学会論文誌 D, Vol.J90-D, No.10, pp.2736–2745 (2007).
[14] KH Coder, available from ⟨http://sourceforge.net/projects/khc/⟩ (accessed 2011-01).
[15] Witten, I.H.: *Data Mining, Second Edition: Practical Machine Learning Tools and Techniques*, 2nd edition, The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann (2005).
[16] Doxygen, available from ⟨http://www.doxygen.org/⟩ (accessed 2011-01).
[17] SQLite, available from ⟨http://www.sqlite.org/⟩ (accessed 2011-01).

海谷 治彦 （正会員）

1994 年東京工業大学博士（工学）取得．現在，信州大学工学部准教授，国立情報学研究所客員准教授．

原 賢一郎

2011 年信州大学修士（工学）取得．

小林 亮太郎

2011 年信州大学学士（工学）取得．

長田 晃

2008 年信州大学博士（工学）取得．

海尻 賢ニ （正会員）

1977 年大阪大学工学博士取得．現在，信州大学工学部教授．