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

信学技報
TECHNICAL REPORT OF IEICE.

活動履歴と過去の推薦状況を考慮した変更支援ツールの試作 (訂正版)

山森　章弘†　　小林　隆志††

† 東京工業大学 工学部 情報工学科　〒 152–8552 東京都目黒区大岡山 2–12–1
†† 東京工業大学 大学院情報理工学研究科 計算工学専攻　〒 152–8552 東京都目黒区大岡山 2–12–1
E-mail: †yamamori@sa.cs.titech.ac.jp, ††tkobaya@cs.titech.ac.jp,

あらまし　本稿では，開発者が次に変更するべき箇所を推薦する変更支援ツールを提案する．提案ツールでは，成果物に対して開発者が過去にどのようにアクセスしたかを記録した活動履歴と過去の推薦状況を考慮し，必要となる変更箇所を推薦する．蓄積した活動履歴を用いた評価実験によって，提案ツールの有用性を議論する．
キーワード　変更支援，アクセス履歴，ソフトウェアリポジトリマイニング，ソフトウェア保守

A change guide tool based on interaction and recommendation history

Akihiro YAMAMORI† and Takashi KOBAYASHI††

† Dept. of Computer Science, School of Engineering, Tokyo Institute of Technology
2–12–1, Ookayama, Meguro-Ku, Tokyo, 152–8552 Japan
†† Dept. of Computer Science, Grad. School of Information Sci. & Eng., Tokyo Institute of Technology
2–12–1, Ookayama, Meguro-Ku, Tokyo, 152–8552 Japan
E-mail: †yamamori@sa.cs.titech.ac.jp, ††tkobaya@cs.titech.ac.jp,

Abstract　In this paper, we propose a prototype of a change guide tool which recommends necessary changes. Our proposed tool calculates candidates based on the recommendation history and interaction history which is an read and write access record of artifacts. We discuss effectiveness of our tool through experiments.
Key words　change guide, interaction history, software repository mining, software maintenance

1. はじめに

ソフトウェアの大規模化に伴い，不具合修正や機能追加といった保守工程において依存関係を解析しながら影響の波及先を特定する労力が増大している．開発者の理解不足によって変更が必要な箇所を見落とすことを防ぐために，開発者の変更支援手法が古くから研究されてきた [1]．プログラムを静的解析することによって，変更の波及箇所を求める手法が存在するが [2]，その解析結果はしばしば膨大な範囲を出力し，実際には影響のない範囲も多数含む．

この問題を解決するために，開発者の開発履歴を用いた変更支援手法が近年盛んに研究されている [3–5]．これは，過去にある開発者がプログラム変更中に行った行動と，別の開発者が別のプログラム変更中に行うであろう行動は類似するという仮定のもと，過去の変更履歴から変更波及範囲を求めるという手法である．

開発履歴を用いた多くの既存手法ではバージョン管理システムのコミット単位での記録を開発者の行動として解析しており，一回のコミット中にプログラムのどの部分をどのような順番で変更したのかなどの情報が含まれていない．また，開発者の行動を解析する際に，開発者が変更に際して何を参照したかという情報は，開発者がプログラムを理解する過程が見えるという点で重要な情報になりうるが，バージョン管理システムはプログラムの変更箇所を記録するのみであり，開発者がどのファイルやメソッドを参照したかを記録できない．

これらの問題に対し，我々は開発者が行った変更行為及び参照行為を記録し，ある変更とその一つ前の変更との間の参照情報を変更コンテキスト情報として用いる変更支援手法を提案してきた [6–8]．これらの先行研究では，ある変更から予測する次の変更箇所は 1 箇所であるというモデルを使用している．しかし，実際には変更による影響が複数箇所に波及する場合があり，先行研究では変更波及した箇所を適切に推薦することができないという問題がある．

本研究では，変更が複数箇所に波及した場合に対応するために，直前に行われた変更に基づく次変更の尤度に加えて 2 つ以上前の変更に基づく次変更の尤度を累積する方法を提案する．この手法では 2 つ以上前の変更に関連付けられているファイルやメソッドが直前の変更に関連付けられているファイルやメ

ソッドと同様に変更されやすいという仮定のもと，2 つ以上前の変更から導出した尤度を累積することにより変更推薦の精度の向上を図る．また，提案手法を用いてリアルタイムに開発者を支援する Eclipse プラグインについて説明する．

先行研究で収集した 15 人分のログデータを使用した評価実験についても説明する．実験の結果，ファイルレベル推薦では 3 つ前の，メソッドレベル推薦では 5 つ前の推薦まで考慮することが最適であり，先行研究の手法と比較してファイルレベル推薦では，nDCG(normalized Discounted Cumulated Gain) が 0.05 から 0.07 程度，メソッドレベル推薦では 0.18 程度上昇し，推薦精度が向上したことを示す．

2. 関 連 研 究

2.1 変更波及解析による変更支援

プログラムを静的解析することによって，変更の波及範囲を特定する方法が提案されている [2]．ソースコードを解析することで，メソッドの呼び出しやクラスの継承，フィールド参照といった依存関係を取得することができる．

しかし，これらの依存関係は膨大な数であり，すべての依存関係が変更の伝搬を引き起こすわけではない．Geipel ら [9] は，クラス間依存関係と変更伝搬との関係のうち，半分以上は全く関係がないこと，また一部の依存関係が大部分の変更伝搬を引き起こすことを報告している．また，静的解析では検出できないような隠れた依存関係も存在する [10].

以上のようにソースコード上の依存関係のみで変更支援を行うことは困難であるということが言われてきた．

2.2 改版履歴をマイニングする手法

静的解析による変更支援手法の限界に対処するために，ソフトウェア成果物に対する変更の履歴を用いて，現在の保守作業での変更支援を実施する研究が行われている．

Gall らは，CVS などのバージョン管理システムに蓄積された改版履歴を分析することで，同時に変更される可能性の高い logical coupling の関係にあるファイルを発見する手法を提案した [3]．Zimmermann らが提案した eROSE [4] は，この logical coupling をメソッド単位で特定し，開発者があるメソッドを変更した際に，次に変更するべきメソッドを提案するツールである．Kagdi らが提案した sqminer [5] は，logical coupling では考慮されていなかった変更の順序を擬似的に求めて，変更支援の誤検出を減少させている．

2.3 操作履歴をマイニングする手法

バージョン管理システムの履歴よりも細かい粒度で開発者が行った活動履歴を蓄積し利用する方法として，操作履歴 (interaction history) [11] をマイニングする手法がある．操作履歴とは，開発者が開発中に行った変更，選択，操作といった操作を指す．

Zou らは操作履歴から，参照の切り替えが頻繁に起こるファイル間に interaction coupling という関係が張られるとし [12]，保守作業のタスクを特徴づけた．Bantelay ら [13] は，リポジトリマイニングと操作履歴の両方を利用して interaction coupling を導出し，それぞれ片方を用いて導出するよりも再現率が上がることを示した．

Kersten らが開発した Mylar（現 Mylyn) [14] は，開発者のプログラム成果物への操作の頻度を用いて，現在の作業に関連している成果物をまとめる手法を提案し，IDE（統合開発環境）を通じてこれを開発者に向けて表示するプラグインを開発した．Mylyn から出力される開発者の行動履歴は，有志によって Bugzilla(注1) 上にアップロードされている．Soh ら [15] はこれを利用し，多くのプログラム変更プロセスでは，開発者のプログラム変更順序が参照的でなく，ランダムな順番で変更を行っていることを示した．

Maalej ら [16] は，複数のデベロッパツールを使う開発活動においてそれらの操作履歴を記録し，次に使うべきツールを推薦して支援する手法を提案している．Roehm ら [17] は，コードの変更履歴や Web 検索履歴，コンパイルエラー履歴を収集し，隠れマルコフモデルによって開発者が実装の問題を解決する段階を表現する方法を提案している．このように，開発者の行った参照の履歴はいくつかの既存研究で利用されているが，ほとんどの既存研究はプログラムの変更箇所を推薦して支援するものではなく，プログラムの理解支援を目的としたものである．

Robbes らは，操作履歴を用いた細粒度の logical coupling を提案している [18]．また，様々な変更予測手法の性能評価を行い，最も最近変更された成果物に基いて次の変更箇所を推薦する手法が最も高い精度を出すことを示した [19].

我々はこの操作履歴を参考にして，成果物に対する変更，参照の履歴を取得し，2 つの変更間に行った参照を変更履歴のコンテキストとして用いることでより変更推薦を行う手法支援が行えると考えた [6,7]．を提案してきた．さらに，変更間の時間的局所性を考慮した改善手法 [8] を提案している．

3. 先行研究: 操作履歴を用いた変更支援手法

3.1 先行研究 [6–8] の概要

本研究では，開発者がソフトウェア成果物を変更することや，参照することを**操作イベント**と呼ぶ．ファイル単位ではエディタが開いてから閉じるまでを 1 回の操作イベントとし，メソッド単位ではメソッドの定義部分にカーソルがある間を 1 回の操作イベントとする．変更を伴った操作イベントを**変更イベント**と呼び，ソフトウェア成果物を見るだけの，変更を伴わない操作イベントを**参照イベント**と呼ぶ．

我々が提案してきた手法では，IDE を拡張することで操作履歴を収集し，以下の手順で変更支援を行う．

（1） 変更支援グラフを作成する．(3.2 節)

（2） 変更支援グラフを元に，次変更を予測する．(3.3 節)

3.2 変更支援グラフの作成

変更支援グラフは，ある成果物の変更から次の変更を推薦するための学習モデルであり，ある成果物の次にどの成果物が変更されたかを表わす有向グラフである．エッジには過去にその順で変更された際のコンテキストの情報が蓄積される．変更支援グラフは以下の手順で作成される．紙面の都合上 (1),(2) の説

(注1)：https://bugs.eclipse.org/bugs/

明は割愛する．詳細は [8] を参照されたい．

（1） 操作履歴のクレンジング処理

（2） 操作履歴から属性付き変更シーケンスを作成

（3） 変更シーケンスをもとに変更支援グラフを作成

手順 (3) では，変更シーケンスを時系列順に走査する．まず，着目する変更イベントからそれ以降の変更イベントまでの発生時刻の差が閾値 t_{fpc} 以下であるものを探す．これらの変更は時間的局所性があるものと判断し，着目する変更イベントから直後の変更イベントだけでなく，それらの変更イベント全てとの間にエッジを作成する．エッジには，着目する変更イベントのコンテキスト情報を付与する．この際，直後の変更から順に影響を弱める．この操作を変更シーケンスの全ての変更イベントに行うことで，変更支援グラフを構成する．

3.3 変更推薦の導出方法

ある変更イベント c が発生した直後の変更推薦を導出することを考える．変更支援グラフから変更イベント c で変更された成果物を表すノードを探し，そのノードを始点とするエッジ集合 E を取得する．その変更において，各エッジ $e \in E$ が次の変更となる尤度を以下の式で定義する．

$$likelihood_e(c) = \sum_{p \in P_e} \{(1-\alpha) + \alpha(contextMatch(p,q)\} \quad (1)$$

ただし，P_e はエッジ e のコンテキスト情報の集合，q は c のコンテキスト情報である．$contextMatch(p,q)$ はコンテキスト情報同士の類似性を求める関数である．α はコンテキスト情報が変更イベントとエッジで一致するかどうかを尤度に加味する度合いであり，この手法のパラメータである．$\alpha = 0$ ならば $contextMatch$ を考慮せず，$\alpha = 1$ ならば $contextMatch$ のみを考慮する．先行研究ではパラメータの調整を行っており，$\alpha = 0.9$ で最も良い評価が得られている．

このようにして尤度を求めたエッジを降順に並べ替え，尤度の最も大きいエッジの終点の表す成果物を次の推薦の第 1 位とし，第 2 位以下も同様に求めて推薦リストを作成する．

3.4 先行研究の問題点

先行研究では，ある変更から予測する次の変更が 1 箇所であると仮定し，直前に発生した変更イベントのみを尤度関数への引数として変更推薦をしている．しかし，現実には開発者はある変更から変更波及した 2 つ以上の変更を行う場合がある．

たとえば，A の変更によって，B と C の双方が修正が必要となるソースコードがあり，過去に類似する保守作業によって，A → B の順および A → C の順で別々に修正作業が行われた場合を考える．別々に行われた変更であるので B → C のエッジは作成されない．(注2)

この場合，A に変更を行った後の変更推薦は B，C となるが，推薦に基づき B，C の順に変更する場合，B を変更した後は B から C へのエッジが存在しないため，C は推薦されず変更漏れとなってしまう．学習が十分でない新規クラスやメソッドなどへの依存がある場合も同様の問題が発生する．

(注2)：先 A → B → C の変更が一定時間内に行われた場合にのみ, 時間的局所性を考慮して A → B，B → C に加え，A → C のエッジが作成される [8].

4. 提 案 手 法

4.1 累積尤度を用いた推薦

本研究では影響が 2 箇所以上に波及し，時間的局所性を考慮しても解決できない場合に対応する手法を提案する．先行研究の手法では，直前の変更イベントを表すノードを変更支援グラフ中から探し，それを始点とするエッジのみを変更推薦の探索対象としていた．しかし，ある影響が 2 箇所以上に波及する場合は 2 つ前以前の変更イベントのノードを始点とするエッジを探索することも重要となる．そこで，本研究では 2 つ前以前の変更イベントも，変更支援グラフのノード探索条件に利用する．

2 つ前以前の変更イベントを利用する場合，直前の変更イベントが最も重要であり，2 つ前以前の変更イベントは直前の変更イベントよりも重要でない. 本研究では，1 つ前の変更イベントから導出した尤度に，2 つ前以前の変更イベントから導出した尤度を適切な重みを掛けながら足し合わせ，最終的な尤度とし，本研究では**累積尤度**と呼ぶ．

累積尤度を用いた次変更候補 v に対する尤度関数 $cumLikelihood_v(c)$ は以下の式で定義される．提案手法では $likelihood_e(c)$ の代わりにこの $cumLikelihood_v(c)$ を用いて次変更の尤度を計算し，尤度の高い順に推薦を行う．

$$cumLikelihood_v(c) = likelihood_v(c) + \sum_{i=1}^{n} weight(i) * likelihood_v(c_i) \quad (2)$$

ただし，c_i を i 個前に発生した変更イベント，n を探索する過去の変更イベントの個数 (注3)，$likelihood_v(c)$ を変更イベント c から導出した次変更候補 v の尤度を返す関数, $weight(i)$ を i 個前に発生した変更イベント c_i の次変更の尤度 $likelihood_v(c_i)$ に対する重みとする．

$likelihood_v(c)$ は，変更イベント c を表すノード v' からノード v へのエッジを e としたとき，先行研究の尤度関数 $likelihood_e(c)$（3.3 節参照）を用いて以下のように表すことができる．

$$likelihood_v(c) = likelihood_e(c) \quad (3)$$

累積される対象は $likelihood_v(c)$ であり，実際の推薦で利用される $cumLikelihood_v(c)$ ではないため，$n > 2$ の場合でも重複して累積されることはない．

4.2 支援ツール：Eclipse プラグイン

先行研究では，提案手法に基づく推薦エンジンは実装されていたが，オフラインでの推薦精度計算が主目的であり，変更支援の機能は実装されていなかった．本研究では，提案手法を実際の開発で利用できるよう，開発者の変更操作に合わせ，リアルタイムに変更推薦を行う Eclipse プラグイン を作成した．本プラグインは，Eclipse プラグインである Mylyn(注4) [14] を拡張して実装しており，以下の機能を有する．

(注3)：変更イベントを時系列順にならべると $\{c_n, c_{n-1}, \cdots, c_1, c\}$ となることに注意されたい．

(注4)：http://www.eclipse.org/mylyn/

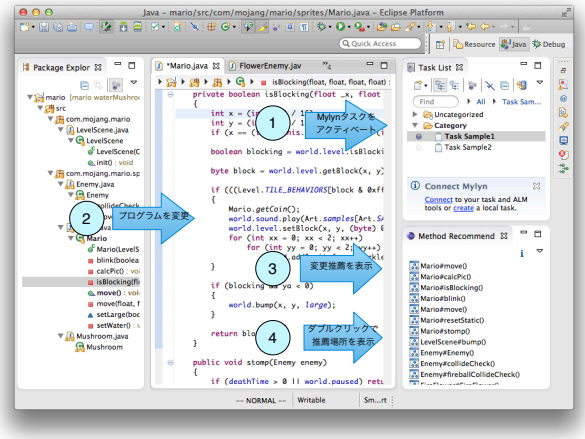

図 1　プラグインのスクリーンショット

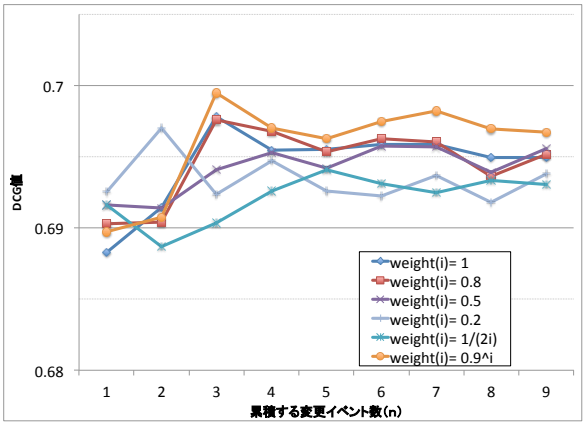

図 2　ファイルレベルでのパラメータの決定

- 開発者の操作イベントを逐次取得する
- 次の変更推薦箇所の算出・表示する
- XML 形式による操作履歴の保存

プラグイン使用時のスクリーンショットを図 1 に示す．開発者は (1) 右上の Task List ビューで Mylyn タスクをアクティベートすることで，推薦機能を有効化する．プラグインは (2)Eclipse 上での操作を監視・記録し，変更が起こるたびに推薦エンジンを用いて (3) 次変更候補を推薦する．開発者は (4) 推薦リストをダブルクリックすることによって推薦箇所へ移動し次の変更を容易に行うことができる．

Mylyn の操作イベントは 8 つの種類があるが，変更イベントと参照イベントの区別がなされていない．本プラグインでは，”org.eclipse.core.filebuffers”パッケージを利用し，操作イベントの前後でファイル・バッファーの内容に変更があるかどうかを調べて，変更イベントと参照イベントを区別する．

また，操作履歴は専用の DB に格納される以外に，XML 形式でも保存される．保存される XML は，Mylyn が保存する context.xml を拡張し，操作イベント以外に，(1) 開発者がダブルクリックした推薦の順位，(2) そのとき表示されていた全推薦の個数，(3) そのときの推薦モード（メソッド，ファイル）の 3 つの情報が記録される．

5. 実　　験

提案手法の効果を確認するために 2 つの実験を行う．まず累積尤度を算出する際の $weight(i)$ および n の影響を検証する実験 1 を行う．次に実験 1 で求めた最適なパラメータを用いて先行研究の手法と推薦精度の比較を行う．

5.1　実験対象の操作履歴と評価基準

推薦精度の評価では，先行研究 [8] で収集した Eclipse 上での操作履歴を用いる．この操作履歴は，スーパーマリオブラザーズの Java 版クローン (注5)(48 クラス，7KLOC) を対象に，学生 15 名がそれぞれ異なるプログラム変更タスクを実行しものである．操作履歴の蓄積には Plog [20] を用いている．

(注5)：https://mojang.com/notch/mario/

手法の精度を測る尺度には，Robbes らの行った変更支援手法の評価 [19] と同様に nDCG (normalized Discounted Cumulated Gain) [21] を利用する．nDCG は，情報検索システムを評価する尺度である．nDCG 値が高いほど，その変更推薦が未来の変更を予測できていると評価することができる．本実験では，各推薦における nDCG を求めそれらの平均である $\overline{nDCG}$ を評価指標とする．

5.2　実験 1: 次変更の尤度の重みの影響

5.2.1　実 験 手 順

各尤度の重み $weight(i)$ と，累積する変更イベントの個数 n を変化させた時のファイルレベルとメソッドレベルでの推薦精度の影響を調査する．n は 1 から 9 までの変化させ，$weight(i)$ の候補は，固定値 4 種と，古い変更イベントほど重みが減衰する関数 2 種を比較した．

- $weight(i) =$ 固定値 $(1, 0.8, 0.5, 0.2)$
- $weight(i) = \frac{1}{2i}$ $(= 0.5, 0.25, 0.166, \dots)$
- $weight(i) = 0.9^i$ $(= 0.9, 0.81, 0.729, \dots)$

5.2.2　実 験 結 果

ファイルレベルでの変更推薦で，各 $weight(i)$ で n を変化させたときの $\overline{nDCG}$ 値のグラフを図 2 に示す．ファイルレベルでは，変更推薦のパラメータが $n = 3, weight(i) = 0.9^i$ のときに $\overline{nDCG}$ が最大になり，この時の $\overline{nDCG}$ は 0.699 であった．また，$n = 0$ のときの $\overline{nDCG}$ は 0.642 であった．

メソッドレベルでの変更推薦で，各 $weight(i)$ で n を変化させたときの $\overline{nDCG}$ のグラフを図 3 に示す．メソッドレベルでは，変更推薦のパラメータが $n = 5, weight(i) = 0.9^i$ のときに $\overline{nDCG}$ が最大になり，この時の $\overline{nDCG}$ は 0.671 であった．また，$n = 0$ のときの $\overline{nDCG}$ は 0.494 であった．

5.3　実験 2: 先行研究との比較

実験 1 で最も $\overline{nDCG}$ が高かったパラメータを用いて，先行研究との比較を行う．3.3 節で述べた α を変化させながら交差検定し，$\overline{nDCG}$ を比較する．

ファイルレベルでの $\overline{nDCG}$ の変化は図 4 のとおり．先行研究の手法と提案手法との比較では，α が 0.9 以下では $\overline{nDCG}$ が約 0.05 から 0.07 程度上がっており，α が 1 の時でも精度が下がらないことが確認できた．

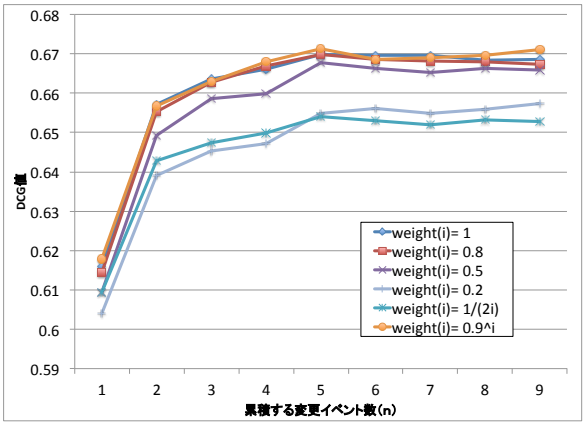

図 3　メソッドレベルでのパラメータの決定

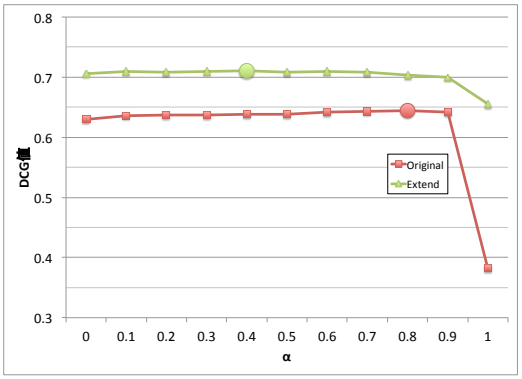

図 4　ファイルレベルでの先行研究の手法と提案手法の比較

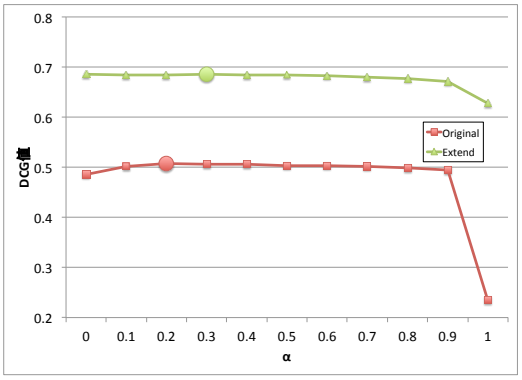

図 5　メソッドレベルでの先行研究の手法と提案手法の比較

メソッドレベルでの $\overline{nDCG}$ の変化は図 5 のとおり．先行研究の手法と提案手法との比較では，ファイルレベルと同様の傾向があり，α が 0.9 以下では，$\overline{nDCG}$ が 0.18 程度上がっていることが確認できた．

5.4　考　　察

5.4.1　実　験　1

ファイルレベル，メソッドレベルともに，$\overline{nDCG}$ の変化のグラフ（図 2, 図 3）は $1 \leqq n \leqq 3$ の区間では概ね右肩上がりである．したがって，3 つ前までの変更イベントを累積することは，重み付け $weight(i)$ にかかわらず推薦精度の向上につながると考える．

また，$\overline{nDCG}$ が最も高くなるパラメータをファイルレベル ($n = 3$, $weight(i) = 0.9^i$) とメソッドレベル ($n = 5$, $weight(i) = 0.9^i$) で比較すると，メソッドレベルのパラメータはファイルレベルよりも n が大きい．メソッドはファイルよりも粒度が細かいことから，開発者が同じメソッドを変更する時間はファイルを変更する時間よりも短くなるため，同じ時間中に発生する変更イベント数はファイルレベルよりも多くなる．以上のような理由から，メソッドレベルの変更推薦ではファイルレベルの変更推薦よりも多くの変更イベントを累積した方が推薦精度が向上したものと考える．

古い変更イベントほど減衰するような重み付けでは，2 つ前以前の変更イベントの重み付けがすべて等しい重み付けよりも $\overline{nDCG}$ が低くなる場合があり，我々の予想に反していた．この結果についてメソッドレベルの操作履歴を詳しく調べたところ，15 個の操作履歴のうち，等しい重み付けのときに $\overline{nDCG}$ が著しく高くなるものが存在した．また，それらの操作履歴は共通して,「メソッド名のリネーム中に，リネーム途中のメソッド名が記録された変更イベントが発生している」という特徴があり，これらの変更イベントが発生したことにより，本来は近い過去に発生していた変更イベントがより遠くなってしまい，結果として $\overline{nDCG}$ が上がっていた．これは，本来操作イベントとして記録されるべきではなく，操作履歴を収集する際の不備であるといえる．

5.4.2　実　験　2

提案手法の $\overline{nDCG}$ が先行研究よりも上がった原因について，評価に用いた操作履歴を読んで検証したところ，4.1 節で説明した利点を含む 3 つの特徴を見つけることができた．

a)　ある変更が 2 箇所以上に変更波及した場合

被験者 A の操作ログの中に，以下の順の変更イベントが記録されていた．

```
1  Fireball#move()
2  Sparkle#move()
3  Sparkle#Sparkle()
4  Fireball#move(float x, float y)
```

Fireball クラスの編集中に Sparkle クラスへの変更が必要になり，短い変更を行ってすぐ元の作業に戻っている．提案手法が対象としている典型的な例である．このなかで先行手法は 1 番から 4 番への変更推薦が可能だったが，それ以外の変更推薦は不可能であった．そのため，この 4 つの変更イベントに対する $\overline{nDCG}$ は低かった．これに対し，提案手法では 3 番の時点で 1,2,3 の累積尤度を用いて推薦を行うため，4 番を 1 位に予測できていた．そのため，$\overline{nDCG}$ は先行研究の手法と比べ大幅に上昇した．

b)　直前の 2 つ以上の変更が共通して推薦できる場合

被験者 A の操作ログの中に，以下の順の変更イベントが記録されていた．

```
1 Fireball#move()
2 Fireball#move(float x,float y)
3 Fireball#Fireball(LevelScene level, float x, float y, int
      facing)
```

2 番から 3 番を変更推薦する時に，先行研究では 3 位で推薦していた．また，1 番から 3 番への変更推薦も先行研究では 3 位であった．

このように 1 番および 2 番が共通して 3 番を推薦していたため，累積尤度を用いた提案手法では 2 番から 3 番への変更推薦は 1 位となった．これにより $\overline{nDCG}$ が上昇した．

c) シグネチャを変更した場合

被験者 B の操作ログの中に，以下の順の変更イベントが記録されていた．

```
1 FireEnemy#FireEnemy(QLevelScene; int x, int y)
2 FireEnemy#FireEnemy(QLevelScene; int x, int y, int a)
3 FireEnemy#FireEnemy(QLevelScene; int x, int y, int a, int b)
4 FireEnemy#move()
```

シグネチャの変更が行われており，推薦エンジンはこれらを別々の変更イベントとして扱っている．そのため先行研究では 1 番から 4 番へのは可能であるが，2 番から 4 番および 3 番から 4 番への変更推薦を行うことが出来ない．

提案手法においては，3 番から 4 番への変更推薦において 1,2,3 番の累積尤度を用いるため，変更推薦が可能である．これにより $\overline{nDCG}$ が上昇した．前節のリネーム中の変更イベントと同様に，本来ならばシグネチャの変更は別々の変更イベントとして扱うべきではない．この点に対する対応方法も今後の課題である．

5.5 妥当性への脅威

a) 操作履歴の妥当性

交差検定に使用した操作履歴は，被験者が変更支援なしで行ったプログラム変更によって収集したものである．そのため，変更支援ありでプログラム変更を行った場合，変更支援なしよりも早く目的の変更箇所にたどり着くことができ，記録される操作履歴が変わってしまうと考えられる．したがって，提案手法の評価は変更支援ありの操作履歴でも行うべきである．

また，実際には開発者がプログラムを間違えて変更してしまった場合でも変更イベントが操作履歴に記録されるが，本論文ではこのような間違った変更と正しい変更を区別せずに推薦制度の評価を行っている．

5.4.1 節で示したとおり，操作履歴には成果物がリネームされている途中で発生した操作イベントも記録されている．これの影響で $\overline{nDCG}$ が向上したケースが存在した．そのため，実験 1 のパラメータの調整は再考の必要がある．

b) シグネチャ変更やソフトウェア成果物の削除

考察（5.4.2 節）でも述べたが，シグネチャの変更やソフトウェア成果物の削除等を推薦エンジンは検知していない．そのため，シグネチャの変更や成果物の削除が行われると，変更済み，消去済みの成果物を推薦し続けてしまい，推薦精度が下がることがある．

c) 統計的妥当性

交差検定では 15 人分の操作履歴ログを用いているが，十分多いとはいえずさらに多くのサンプルを集める必要がある．

6. おわりに

操作履歴を用いて変更推薦を行う先行研究において，ある変更が 2 箇所以上に変更波及した場合に推薦を続けられない問題に対応するため，累積尤度を用いてこの問題を解決し，変更推薦の精度向上を図った．また，Eclipse 上で動作し，推薦箇所の提示や操作履歴の出力等を行うプラグインを作成した．

提案手法によって先行研究で収集した 15 人分の操作履歴ログを交差検定したところ，先行研究の手法による変更推薦よりも nDCG 値が上昇し，累積尤度を用いた手法の効果を確認できた．

謝辞 本研究の一部は科研費 (24300006, 25730037) の助成による．

文　献

[1] R.S. Arnold and S.A. Bohner, “Impact analysis-towards a framework for comparison,” Proc. ICSM’93, pp.292–301, 1993.
[2] L.C. Briand, J. Wust, and H. Lounis, “Using coupling measurement for impact analysis in object-oriented systems,” Proc. ICSM’99, pp.475–482, 1999.
[3] H. Gall, K. Hajek, and M. Jazayeri, “Detection of logical coupling based on product release history,” Proc. ICSM’98, pp.190–198, 1998.
[4] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller, “Mining version histories to guide software changes,” IEEE TSE, vol.31, no.6, pp.429–445, 2005.
[5] H. Kagdi, S. Yusuf, and J.I. Maletic, “Mining sequences of changed-files from version histories,” Proc. MSR2006, pp.47–53, 2006.
[6] 加藤，小林，阿草，“変更支援のための成果物アクセス履歴マイニング,” 信学技報 (ソフトウェアサイエンス)，第 110 巻，pp.145–150，SS2010-77，Feb. 2011.
[7] T. Kobayashi, N. Kato, and K. Agusa, “Interaction histories mining for software change guide,” Proc. RSSE2012, pp.73–77, 2012.
[8] 丸岡，小林，阿草，“成果物アクセスの時間的局所性を考慮した変更コンテキストモデル,” 信学技報 (ソフトウェアサイエンス)，第 112 巻，pp.97–102，SS2012-76，Mar. 2013.
[9] M.M. Geipel and F. Schweitzer, “Software change dynamics: Evidence from 35 java projects,” Proc. FSE2009, pp.269–272, 2009.
[10] G. Canfora, M. Ceccarelli, L. Cerulo, and M. Di Penta, “Using multivariate time series and association rules to detect logical change coupling: An empirical study,” Proc. ICSM2010, pp.1–10, 2010.
[11] W.C. Hill, J.D. Hollan, D. Wroblewski, and T. McCandless, “Edit wear and read wear,” Proc. CHI’92, pp.3–9, 1992.
[12] L. Zou, M.W. Godfrey, and A.E. Hassan, “Detecting interaction coupling from task interaction histories,” Proc. ICPC2007, pp.135–144, 2007.
[13] F. Bantelay, M.B. Zanjani, and H. Kagdi, “Comparing and combining evolutionary couplings from interactions and commits,” Proc. WCRE2013, pp.311–320, 2013.
[14] M. Kersten and G.C. Murphy, “Mylar: a degree-of-interest model for ides,” Proc. AOSD2005, pp.159–168, 2005.
[15] Z. Soh, F. Khomh, Y.-G. Guéhéneuc, G. Antoniol, and B. Adams, “On the effect of program exploration on maintenance tasks,” Proc. WCRE2013, pp.391–400, 2013.
[16] W. Maalej and A. Sahm, “Assisting engineers in switching artifacts by using task semantic and interaction history,” Proc. RSSE2010, pp.59–63, 2010.
[17] T. Roehm and W. Maalej, “Automatically detecting developer activities and problems in software development work,” Proc. ICSE2012, pp.1261–1264, 2012.
[18] R. Robbes, D. Pollet, and M. Lanza, “Logical coupling based on fine-grained change information,” Proc. WCRE2008, pp.42–46, 2008.
[19] R. Robbes, D. Pollet, and M. Lanza, “Replaying ide interactions to evaluate and improve change prediction approaches,” Proc. MSR2010, pp.161–170, 2010.
[20] 谷，小林，山本，阿草，“スタックトレース情報を用いた問題解決経験の検索,” FOSE2008 論文集，pp.99–104，2008.
[21] K. Järvelin and J. Kekäläinen, “Cumulated gain-based evaluation of ir techniques,” ACM TOIS, vol.20, no.4, pp.422–446, 2002.