プロセスマイニングを用いたソースコード分割

加藤光幾[†1]　金井剛[†1]

業務アプリケーションに対する新規機能追加などの保守は低コストで行うことが望まれる．そのため今後機能追加が予想される業務のまとまり（業務単位）を優先してリファクタリングを行い，機能を追加しやすい状態に保つことが望ましい．

本稿では，我々が開発したプロセスマイニングツールである BPM-E を用い，それが業務データから抽出する業務フローとその実行量を基に，業務単位とそれらに対応するソースコードの対応を取得する方法を提案する．本技術を社内の業務システムに適用した結果，イベントのグループ分割が適切に行われることを確認できた．一方イベントとソースコードの対応付けで精度に課題があることが判明した．

Source Code Partitioning Using Process Mining

Koki Kato and Tsuyoshi Kanai

Software maintenance of business application software such as adding new functions should be performed cost-effectively. It is preferable to give priority to perform refactoring the portion of source code to which the function addition will be expected in the future to keep the maintainability.

We propose a new method which extracts such portion of source code using the BPM-E process mining tool we have developed. The method was applied to in-house business systems; the results showed that the method successfully extracted the grouping of events, but that there are accuracy issues in associating events with source code.

1. はじめに

企業で日常の業務に用いられる業務アプリケーションには，顧客ニーズの変化や企業戦略の変更に対応するため，常に機能追加が要求され，新しい業務機能を追加するための適応保守が発生する．長年に渡り使用されている業務アプリケーションには，開発時の工数よりも適応保守の工数の方が多く投入されている[3]．保守予算には限りがあるため，予算を有効に使うためには機能追加要求に対して適切な優先度付けが必要となる．

業務アプリケーションが長期に使われていくと，ソフトウェアの保守がより困難になる傾向がある[3]．このような老化を防止する手段として，リファクタリングや再構築のような完全化保守が有効である．完全化保守を行うにはコスト（工数）が発生するが，適用後には保守性が向上し[2]，機能追加をより低コストで行えることが期待される．リファクタリングなどを業務アプリケーション全体に行うのは高コストになるため困難であり，今後機能追加が予想される業務単位を優先して行うのが望ましいと考える．ここで言う業務単位とは，業務の最小単位（アクティビティ）がいくつかまとまって実行される単位で，複数の業務で共有されることもあるものを意味することとする．実行量が増加傾向にある業務単位は業務的に成長しており，機能追加が期待される箇所と考えることができる．そのため，実運用における業務単位を抽出し，それを実行しているソースコードと対応づける技術が必要である．

上記のような実運用での業務の動きを把握する技術として，プロセスマイニング（process mining）技術[4] が知られている．プロセスマイニングは業務実行時に業務システムから発生するイベントログなどを用いて，業務を動的な観点から分析する．イベント（アクティビティなどとも呼ばれ，プロセスマイニングで把握する業務の最小単位）の種別やイベントの実行順，それらの実行量，実行時間などを抽出することができる．

本稿では業務アプリケーションへの機能追加に役立つ情報を提供するために，プロセスマイニングツールの一つである BPM-E で得られる業務フローと実行量情報を用い，業務単位とそれらを実行するソースコードの対応を抽出する方法を提案する．以下では BPM-E の機能概要を説明し，業務単位の抽出とイベント種別に対応するソースコードの抽出方法を述べ，本方式を 2 つの社内システムに適用した結果と考察を示す．

2. BPM-E

Business Process Management by Evidence （BPM-E; 北米では Automated Process Discovery (APD) という名称を用いている）[7] は富士通研究所が開発したプロセスマイニングツールの一種であり，既にアメリカ，イギリス，日本などで，顧客業務を分析する手段のひとつとして使用されている．

BPM-E の特徴のひとつは，CSV 形式で入手した顧客業務システム内の業務テーブルデー

†1 (株) 富士通研究所ソフトウェアシステム研究所
Software Systems Laboratories, Fujitsu Laboratories Ltd., Japan

タ一式をそのまま用いて業務フローを抽出・分析することである．多くの顧客は業務システムに手を加えることにきわめて慎重で，ログ出力機能の追加は困難である．一方，業務システム内データのバックアップは通常業務で行っておりデータを CSV 形式で出力するのは容易なので，システムの変更なしに短工程で BPM-E を適用できる．

2.1　イベントと業務フロー

BPM-E では業務テーブルに記録されている業務的な出来事をイベントと称する．イベントは（ID，タイプ，時刻）の組で表現される．ID は業務上の識別子（たとえば受注番号）であり，タイプはイベントの種別（たとえば「受注」や「発送」）であり，時刻はイベントが発生した時刻である．

ID でイベントをグループ化しイベントの時刻順に並べたイベントの列，例えばある ID で発生した “受注→在庫確認→梱包→発送” のようなイベント列をフローインスタンス，その種類をフロー種別と称する．フロー種別ごとの実行量や平均リードタイム，あるイベント種別からあるイベント種別への平均遷移時間/偏差/遷移回数，通常の業務と異なる実行順であったフローインスタンスの検出・表示などを用いて，業務プロセスの問題を分析する．

2.2　BPM-E が処理対象とする業務テーブル

業務システムで用いられるテーブルには大別してマスターデータテーブル（部品や商品などの記号と説明などの一覧）と，業務テーブル（日頃の業務オペレーションによって発生する ID や発生時刻の情報）があるが，BPM-E は業務テーブルを分析対象とする．

業務テーブルは，作業履歴を記録する形態として，次のタイプが存在する．

- ログ・履歴型: イベントが発生するごとに，どのイベントの種別が発生したかを表す情報と時刻と ID を 1 レコードとして記録する．
- イベント種別を格納するカラムが存在しない: イベント種別は次のように識別できる．
  - テーブル自体がイベント種別に対応: イベントはイベント種別に対応した専用のテーブルに記録される．イベントの種別はテーブル名で識別することが出来る．
  - テーブルのカラム名がイベント種別に対応: ひとつのテーブルに複数のタイムスタンプのカラムが存在し，それぞれが異なるイベント種別に対応する．あるイベントが発生すると，対応するイベント種別のカラムにタイムスタンプを書き込む．
- レポート用一覧テーブル: 上記の「テーブルのカラム名がイベント種別に対応」に類似しているが，他のテーブルの時刻情報の複製であり進捗把握に使用する．

BPM-E は上記タイプの組み合わせにも対応可能である．

BPM-E の主な分析機能には，業務フロー図，例外フロー表示，絞り込み検索，フロー種別一覧，フローの比較，などがある．

3.　ソースコードの分割手法

以下では Java と SQL で記述された業務アプリケーションを対象とする．

例えば受発注システムでは「受注」アクティビティが単独では発生せず「在庫確認」などと組になって実行されるかもしれない．このような挙動はソースコードの静的分析では分からず，プロセスマイニングのような動的分析によって抽出可能である．

一方，アクティビティ実行時には Java メソッドで SQL 文（`INSERT` 文ないし `UPDATE` 文）に ID や時刻などのパラメータを設定・実行し，データベースのテーブルに記録する動作が発生する．テーブルに記録された値は，BPM-E で 2.2 に示したようにイベントとして抽出されるので，逆にそのイベントを生成可能な SQL 文を持つメソッドを検索しそのメソッドの呼び出し関係を辿ることで，イベント発生時に実行されるメソッドの集合が推定できる．つまり業務単位を構成するイベント種別が分かれば業務単位に対応するソースコード（メソッド）が推定できる．以下に業務単位の抽出と，ソースコードとの対応付けを述べる．

3.1　業務単位の抽出

フロー種別を観察すると，フロー種別にほぼ共通して発生する業務単位と，フロー種別を特徴付ける業務単位があることが分かる．前者はたとえば受発注システムでの「受注」などであり，後者は在庫不足時のみに起こる「製造依頼」などである．そこで次のように業務単位を分類する．

- 共通イベント（G_0）: フロー種別にほぼ共通して発生するイベント種別の集合．
- （その他の）業務単位（$G_1, G_2, \ldots$）: 共通イベントを除いたイベント種別のうち，業務特有のイベント種別集合．

業務単位抽出は次の手順で行う: (1) 最初に共通イベントを抽出する，(2) 残りのイベントを対象に，実行量の多いフロー種別の順にイベント種別のグループを作成する，(3) 得られた結果に対して補正処置をする．

以下にそれぞれの説明を示す．

3.1.1　共通イベントの抽出

共通イベント G_0 は，フロー種別の実行量にかかわらず，あるイベント種別を含むフロー種別の割合が閾値（τ_c）以上であるようなイベント種別の集合として求める．

3.1.2　共通イベントを除く業務単位の抽出

フロー種別のうち，実行量が多い方がその企業でより標準的に実行され重要であると考え

```
P ← G_0
for i = 1 to M do
  G_i ← {}
  for j = 1 to n_i do
    if E_{i_j} ∉ P then
      G_i ← G_i + E_{i_j}
      P ← P + E_{i_j}
    end if
  end for
end for
```

(a) 業務単位の抽出

注:

M: フロー種別の個数

$F_i = (E_{i_1}, E_{i_2}, \ldots, E_{i_{n_i}})$: i 番目に実行量の多いフロー種別

E_{i_j}: フロー種別 F_i で j 番目に発生したイベント種別

```
for j = M downto 2 do
  for i = 1 to M do
    C_i ← 0 { カウンタのクリア }
  end for
  for i = 1 to M do
    if F_i が G_j のイベント種別をすべて含む then
      for k = 1 to j − 1 do
        if F_i が G_k のイベント種別をすべて含む then
          C_k ← C_k + #(F_i)
        end if
      end for
    end if
  end for
  C_total ← Σ_{i=1}^{M} C_i
  C_max ← max(C_1, ..., C_M)
  maxPos ← x, where C_max = C_x
  if C_max/C_total > τ_m then
    Gmove.put(j, maxPos)
  end if
end for
for j = M downto 1 do
  maxPos ← Gmove.get(j)
  if maxPos ≠ null then
    G_maxPos ← G_maxPos + G_j
    G_j ← {}
  end if
end for
```

(b) 補正処置

図 1　業務単位抽出アルゴリズム
Fig. 1　Algorithm of extracting work packages

られる．そのため，業務単位を抽出する際には，実行量が多いフロー種別に含まれるイベント種別を優先するのが妥当と考える．そこで，フロー種別の実行量の多い順に，まだ分類されていないイベント種別があればそれらを新しい業務単位として登録する．共通イベントとして抽出されなかったイベント種別から業務単位を抽出する擬似コードを**図 1** (a) に示す．図中，P はすでに処理したイベント種別を格納するための変数である．

3.1.3　補 正 処 置

上記で得られる業務単位は，他の業務単位と併合してより大きな業務単位として捉えるべきものを含む．そのため，最後に加えた業務単位から順に，あるフロー種別が業務単位を含むとき，同時に他の業務単位を含む率が閾値 τ_m 以上の場合にそれらをマージする処理を行う．擬似コードを図 1 (b) に示す．図中，'Gmove' は連想記憶で，$Gmove.put(i, j)$ は i に対して j を記憶し，$Gmove.get(i)$ は j を返すものとする．また，C_k は $k < j$ のときに G_k と G_j の両方を含むフロー種別の実行量（#() で表している）の合計であり，C_{max}/C_{total} は G_{maxPos} と G_j の両方がどのくらい同時にフローインスタンスに含まれていて G_{maxPos} が G_j にマージできるかを示す．

3.2　イベント種別に直接対応するソースコードの推定

イベント種別と対応づけたい Java メソッドは `INSERT` や `UPDATE` 文に ID やタイムスタンプなどのパラメータを設定するメソッドである．引数として渡される SQL 文を実行するメソッドは，直接的にはイベント種別と対応づけられない．

業務単位に含まれるイベント種別と直接対応付くメソッドは次のように抽出する:

(1)　ソースコードから，`INSERT` 文ないし `UPDATE` 文（$SQLstatement$）と，それが記述されているメソッド m からなる組 $(SQLstatement, m)$ を抽出する．

(2)　BPM-E のイベント抽出時の定義情報から，イベント種別 E_j に対応するテーブル名，タイムスタンプのカラム名，イベント種別のカラム名を読み出す．

(3)　E_j が業務単位 G_i に含まれるとき，E_j に対応する $SQLstatement$ に対して，対応するメソッド m を G_i に対応する Java のメソッドの集合 M_{G_i} に格納する．なお，あるメソッドが複数のイベント種別に対応する場合があり得る．

3.3　関連するソースコードの範囲の決定

業務単位に対応するメソッドの呼び出し元/呼び出し先関係の追跡は次のように行う．なお，ユーティリティのように複数箇所から呼び出されるメソッドは共通メソッドとして別扱いする．

(1)　業務単位 G_i に対応するメソッドの集合 $M_{G_i}, i = 0, 1, \ldots$ に対して，M_{G_i} に含まれるメソッド m を呼び出すメソッドを再帰的に検索する．見つかったメソッドを M_{G_i} に登録する．すでに登録されているメソッドに到達したら，検索を止める．

(2)　$M_{G_i}, i = 0, 1, \ldots$ に対して，M_{G_i} に含まれるメソッド m から呼ばれるメソッドを再帰的に検索し，新たに見つかったメソッド m' と業務単位の対を保存する．

(3)　もし複数の業務単位がメソッド m' に対応する場合はメソッド m' を共通メソッド CM に登録し，そうでない場合はメソッド m' を対応する業務単位 $M_{G'_i}$ に登録する．

(4)　登録されなかったメソッド群を，業務とは直接関係のないメソッドとして記録する．

表 1　システム構成
Table 1　System features

	システム A	システム B
ソフトウェアアーキテクチャ	3 層構造+バッチ	3 層構造
フレームワーク	(未使用)	Apache Struts + DbUtils
総クラス数	577	169
総メソッド数	6872	3219
総 LOC (Lines of code) 数	129744	31817
イベント種別数	36	7
フロー種別数	6810	11
業務実行数	307035	1709
開発開始年	2000	2008

表 2　システム A での業務単位抽出結果
Table 2　Results of work package extraction for System A

業務単位	イベント種別
G_0	e1, e2, e3, e4, e5, e6
G_1	e7, e8, e9, e10, e11, e12, e13, e14, e15, e16, e17, e18
G_2	e19, e20, e21, e22, e23
G_3	e24, e25, e26, e27, e28, e29, e30, e31
G_4	e32
G_5	e33
G_6	e34
G_7	e35
G_8	e36

4. 実 験 結 果

社内の業務システム 2 種類を使用して実験した．両者とも業務アプリケーションが Java および SQL で記述されている．ストアドプロシージャは用いていない．特性を**表 1** に示す．Java ソースコードの解析には Eclipse の AST (Abstract Syntax Tree) を使用した．

4.1　業務単位抽出結果

業務単位抽出はシステム A のみで行った．閾値 τ_c と τ_m は今回の実験では 0.8 に設定した．ただし最適値であるかの検証は行っていない．

業務単位とそれを構成するイベント種別の抽出結果を**表 2** に示す．イベント種別名は公表できないため，e1，e2 などに置換している．共通イベントも含め計 9 個の業務単位が抽出された．

4.2　業務単位とソースコードの関連付け結果

システム A において，それぞれの業務単位に対応づけられたメソッド数を**表 3** (a) に示す．複数のイベント種別に対応づくメソッドがあるため，(a) では 27 個の重複したメソッドが存在する．同様にシステム B では表 3 (b) において 2 個の重複メソッドが存在する．

表 3　業務単位に対応するメソッド
Table 3　Results of methods associated with work packages

(a) システム A

業務単位	メソッド数
(共通メソッド)	2600
G_0	3175
G_1	37
G_2	14
G_3	197
G_4	4
G_5	13
G_6	5
G_7	7
G_8	10
(対応しないメソッド)	837

(b) システム B

業務単位	メソッド数
(共通メソッド)	584
G_0	0
G_1	98
G_2	1
G_3	3
(対応しないメソッド)	2535

5. 考　　察

本節では，まず業務単位抽出結果について，業務担当者が記述した業務フロー図との比較より考察する．次に，イベント種別とメソッドの関係付け精度に関して考察する．

5.1　業務単位抽出の評価

提案方式を評価するため，システム A の業務担当者が記述した，4 種類のフロー種別が含まれる業務フロー図から手作業で抽出した業務単位と比較する．

5.1.1　システム A の業務フロー図から得られた業務単位

業務担当者が記述したフロー種別はそれぞれ完全に独立ではなく，アクティビティ（イベント種別）のかたまりで構成される業務単位をいくつか組み合わせて構成されていた．著者の一人は，フロー種別の部分的な類似性に基づき，業務フロー図から業務単位を手動で抽出した．この際，図中で距離が離れて配置されているアクティビティ間の類似性は考慮しなかった．業務単位と，業務単位を用いて業務フローを表現した結果を，それぞれ**表 4** と**表 5** に示す．表中，右矢印は業務単位の実行順を示す．

担当者が記述した業務フロー図の実行量を推定するために，業務フロー図を BPM-E のフロー種別と比較した．比較対象のフロー種別としては，実行量が上位 40（フローのイン

表 4　システム A で手作業で抽出した業務単位と対応するイベント種別
Table 4　Manually extracted work packages and corresponding events for System A

業務単位	対応するイベント種別
Γ1	$e4, e5, e6$
Γ2	$e14, e15$
Γ3	$e2, e7, e8, e9, e10, e12, e13, e17, e18$
Γ4	$e1$
Γ5	$e11$
Γ6	$e26, e29$
Γ7	$e20, e21, e23, e28, e32, e33, e36$
Γ8	$e27$
Γ9	$e34$
Γ10	$e3, e25, e30, e31, e35$
Γ11	$e24$
Γ12	$e19$
Γ13	$e22$
Γ14	$e16$

表 5　フロー種別，業務単位と実行量
Table 5　Flow type, work packages, and number of executions

フロー種別	業務単位によるフロー表現	実行量
タイプ A	Γ1 → Γ2 → Γ3 → Γ4 → Γ5	79697
タイプ B	Γ1 → Γ6 → Γ7 → Γ8 → Γ9 → Γ10 → Γ2 → Γ3 → Γ4 → Γ5	23564
タイプ C	Γ1 → Γ9 → Γ11 → Γ10 → Γ4 → Γ6 → Γ12 → Γ8	11599
タイプ D	Γ13 → Γ1 → Γ7 → Γ3 → Γ4 → Γ12	11522
(その他)		92373

スタンス総数は 218755）のフロー種別を用い，それぞれに含まれるイベント種別からタイプを判断した．実行量を表 5 に示す．なお，担当者が記述した業務フロー図に出現するアクティビティと BPM-E のイベント種別とは必ずしも直接的に 1:1 に対応するとは限らず，1 つのアクティビティが複数のイベント種別に対応する場合やイベント種別に対応するアクティビティが存在しない場合がある．

表 5 より，タイプ A がもっとも多く実行された業務であることが分かる．さらに，その他に分類したものを調査したところ，その大部分は 2 種類のタイプ A の変形であることが分かった．1 種類は Γ14 を含むもので，もう 1 種類は Γ12 を含むものである．これらは，業務担当者による業務記述が網羅的でなかったためと考えられる．

5.1.2　業務単位の分析

表 5 より，タイプ A〜D で共通の業務単位は $\{\Gamma1, \Gamma4\} = \{e1, e4, e5, e6\}$ である．前述の結果と比較すると表 2 の共通イベント（G_0）には余計なイベント種別 $\{e2, e3\}$ が含まれていることが分かる．$e2$ に関しては，$e2$ が Γ3 に含まれておりタイプ A とその変形であるその他の実行量が他のフロー種別よりも多いことを考えると，妥当な結果と考えられる．

一方 $e3$ に関しては，実行量を調査したフロー種別上位 40 個のうち 13 個にしか出現せず，上位 40 個においては共通とは言えない．しかしフロー種別全体を見ると，$e3$ を含むフロー種別は 5541 個であり，フロー種別全体が 6810 個なので 5541/6810=0.81 で閾値とした 0.8 を超えている．BPM-E はデフォルトの場合，イベント種別の順番やあるイベント種別の連続発生回数が異なるフロー種別を区別して扱うため，そのようなバリエーションが多く発生するフロー種別の場合，上記のような現象が発生する可能性がある．

表 6　手作業で抽出した業務単位を用いて業務単位を表現した結果
Table 6　Work packages expressed by manually extracted work packages

G_0	$\Gamma1 + \Gamma4 + (e2) + (e3)$
G_1	$\Gamma2 + (\Gamma3 - e2) + \Gamma5 + \Gamma14 \approx$ タイプ A と B の一部分
G_2	$\Gamma12 + \Gamma13 + (e20 + e21 + e23) \approx$ タイプ D の一部分
G_3	$\Gamma6 + \Gamma8 + \Gamma11 + (e25 + e28 + e30 + e31) \approx$ タイプ C の一部分
G_4	(対応なし)
G_5	(対応なし)
G_6	Γ9
G_7	(対応なし)
G_8	(対応なし)

表 4 に示した手作業で抽出した業務単位を用いて，表 2 に示した業務単位を表現した結果を**表 6** に示す．表 6 より，業務単位 G_0〜G_3 は担当者が記述した業務フローの部分を正しく表現していると考えられる．一方，G_4〜G_8 は業務と関連づけた説明が困難である．

5.2　イベント種別とメソッドの対応付けの評価

ここでは潜在的なカバー率（すべてのメソッドに対してイベント種別に対応するメソッドが占める割合）に関して評価を行う．理想的なカバー率は 1，すなわちすべてのメソッドがイベント種別に対応付く場合である．カバー率が低いことは，実行量を推定できるメソッドが少ないことを意味する．本実験では，システム A のカバー率は 0.88 であったのに対し，システム B は 0.21 であり，システムによって大きな差が生じた．

そこで，イベント種別に対応付かなかったメソッドに傾向があるかを調べるため，Java の package 名によって分類した結果を次に示す．

- システム A: バッチ処理, beans（イベントとは関係しない分）, データベース, servlet, ユーティリティ, CORBA. 特定の package やクラスへの集中は観測されなかった．

- システム B: マスタ保守, 業務ロジック, メニュー, ユーティリティ.

システム B において，Apache Struts （Action と Form）とデータ処理の観点から分類したときのメソッド数は次の通りであった.

- Struts Action + Form [マスタ保守: 89, 業務ロジック: 1648, メニュー: 520]
- データ処理 （データベースアクセスを含む） [マスタ保守: 95, 業務ロジック: 159, ユーティリティ: 23]

上記より，Struts Action と Form に関するメソッドにイベント種別と対応付かないものが多いことが分かる．これは Struts において呼び出し関係が直接的でないためであり，予想されたことではあるが未対応であった．またマスタデータテーブルに関するメソッドはイベント種別と関係付かない．また，今回の実験では `SELECT` 文を対象外にしているため，検索系の処理部分で対応付かない場合があると考えられる．これらの理由により，特にシステム B においてカバー率が低くなったと考えられる.

これらに関してカバー率を向上する方法がいくつか考えられる:

- Struts など定義ファイルで振る舞いを定義するフレームワークを使用する場合に，その定義ファイルも解析して，ページアクセス時のメソッド間の関係を抽出する．ただしフレームワークごとに解析を対応する必要がある.
- `INSERT`/`UPDATE` で書き込んだデータを読み出す `SELECT` 文を分析対象に加える.

なお，今回の実装では動的生成した SQL は未対応である．両方のシステムとも，for ループを用いた SQL 文生成や，if 文で生成する SQL 文を変えるなどの動的処理が含まれている．動的生成に対処するためには，Java のコードを部分的に実行するためのインタプリタや仮想コード実行[1)] などの技術が必要である.

6. 関 連 研 究

実行量の把握とソースコードの実行箇所を直接把握する方法として，アプリケーション実行時のトレース情報や profiling を用いる方式が考えられる．6) では ‘phase detection’ という手法でトレースログを分割し，分割したトレースログを要約し可視化する方法が提案されている．phase detection で抽出された phase は本稿の業務単位に類似した概念と考えられる．しかし，トレースログや profiling を取得するためには業務システムの設定を変更する必要があり，また，実行トレースの取得中は業務システムに無視できない負荷を与えるため，実行トレースを実運用状態にあるシステムから長期に取得するのは困難である．また，テスト環境とテストケースを用いることも考えられるが，たとえテストケースが網羅的であっても実行量が実運用状態とかけ離れているため，実業務での使われ方が把握できない.

一方，BPM-E のようなプロセスマイニングは業務システムに負荷をかけずに実運用状態を把握できるため，精度は実行トレースに劣るものの実用的であると言える.

プロセスマイニングを使用した業務単位のようなグループを抽出する研究として，5) では異なるレベルの抽象度でイベント種別を記述できる方法を提案している.

7. ま と め

本稿では業務単位とその実行量，および関連するソースコードの抽出をプロセスマイニングを用いて行う方法を提案した．これらの情報を用いることで，業務単位でより適切な保守戦略を立てることができるようになる.

本方式を社内の 2 つの業務システムで評価し，業務単位の抽出を確認した．一方，動的に生成する SQL 文や bean の値取得に起因するイベント種別とコードの対応付けの課題に対して，(文献 1) のような）仮想的な実行を SQL 文生成部分に適用できるかを検討したい.

参 考 文 献

1) Brat, G., Havelund, K., Park, S. and Visser, W.: Java PathFinder - Second Generation of a Java Model Checker, *In Proc. of the Workshop on Advances in Verification* (2000).
2) Carriere, J., Kazman, R. and Ozkaya, I.: A cost-benefit framework for making architectural decisions in a business context, *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE ’10, New York, NY, USA, ACM, pp.149–157 (2010).
3) Jones, C.: Geriatric Issues of Aging Software, *CrossTalk*, Vol.20, No.12, pp.4–8 (2007).
4) vander Aalst, W. M.P., Reijers, H.A., Weijters, A. J. M.M., van Dongen, B.F., de Medeiros, A. K.A., Song, M. and Verbeek, H. M. W.E.: Business process mining: An industrial application, *Inf. Syst.*, Vol.32, No.5, pp.713–732 (2007).
5) van Dongen, B.F. and Adriansyah, A.: Process Mining: Fuzzy Clustering and Performance Visualization, *BPM Workshops*, pp.158–169 (2009).
6) Watanabe, Y., Ishio, T. and Inoue, K.: Feature-level phase detection for execution trace using object cache, *Proceedings of the 2008 international workshop on dynamic analysis*, WODA ’08, New York, NY, USA, ACM, pp.8–14 (2008).
7) 富士通：Automated Process Discovery, 富士通 (online), available from ⟨http://www.fujitsu.com/global/services/software/interstage/solutions/bpm/apd.html⟩ (accessed 2011-04-15).