メトリクスと機械学習によるデザインパターン検出

Design Pattern Detection by Metrics and Machine Learning

内山 諭[*]　久保 淳人[†]　鷲崎 弘宜[‡]　深澤 良彰[§]

あらまし 既存のオブジェクト指向プログラムからデザインパターンを自動検出することで，プログラムの理解を助け，保守性や再利用性を向上させる試みがある．従来の検出手法の多くは静的解析に基づくため，クラス構造が類似したデザインパターンの検出が困難である．さらに，クラス構造に関する厳格な条件記述を用いて検出するため，デザインパターン適用の多様性への対応が難しい．我々は，メトリクスと機械学習によるデザインパターン検出手法を提案する．本手法は，メトリクスの測定値を入力とした機械学習を用いてデザインパターンを構成するロール候補を判定し，ロール候補間の関連からデザインパターンを検出する．本手法により従来の検出手法の問題解決を達成する．

1　はじめに

デザインパターン（以下，パターン）とはオブジェクト指向ソフトウェア開発における設計工程で繰り返し現れる問題に対する解法, 指針である [1]．パターンは，ある関連の上で果たしている役割，立場，能力を表すロールを持つクラスによって構成される．代表的なパターンとして，GoF パターン [1] が挙げられる．例えば，図 1 は GoF パターンの 1 つである State パターンである．State パターンは Context，State，ConcreteState といったロールを担うクラスから構成される．パターンの適用によって再利用性や保守性の高いソフトウェア開発，プログラムの理解性の向上，開発者同士のスムーズな意思疎通をもたらすことができる．

オープンソースや第三者が作成したプログラムの読解は時間コストがかかり，既存のプログラムの中にはクラス名やコメント，あるいは付属文書などに明記されずにパターンが適用されている場合もあるため，パターンの検出によってプログラムの理解性が向上する．しかし，既存のプログラムからパターンの適用箇所を人手で検出するには，多大な労力やコストを費やし非効率的である．また，人手による検出は属人性が高いため検出の見落としが発生する．

上記の問題の解決策として，パターンの自動検出に関する研究がされている．従来の検出手法の多くは静的解析を用いた検出手法である．しかし，これらの手法には，クラス構造が類似したパターンや特徴の少ないパターンの検出が困難であるという問題がある．さらに，クラス構造に関する厳格な条件記述を用いて検出するため，少しでもその条件記述から外れた場合に検出の見落としとなる可能性がある．

我々は，メトリクスと機械学習によるパターン検出手法（以下，本手法）を提案する．本手法は検出手法の中でも静的解析に分類できるが，従来の検出手法のような厳格な条件記述（静的構造など）を用いるのではなく，メトリクスの測定値を入力とした機械学習によって導出された判別要素からパターンを検出する．メトリクスとは，ソフトウェア開発をさまざまな視点から評価するための定量的尺度 “メトリック（Metric）” の集合を指し，例としてクラス中のメソッド数の測定などが挙げられる [2]．本手法では，メトリクスを用いることで，それらの組み合わせから未知の判別要素を導出する可能性がある．また，検出結果が機械学習の学習データの種

[*]Satoru Uchiyama, 早稲田大学

[†]Atsuto Kubo, 国立情報学研究所

[‡]Hironori Washizaki, 早稲田大学

[§]Yoshiaki Fukazawa, 早稲田大学

類や数に依存するため，パターンの適用形態に関する様々な種類の学習データを多く用意することで，パターン適用の多様性に対して網羅的に検出できると考えられる．

本稿の構成は以下のとおりである．第 2 章では従来の検出手法の紹介や問題点を含めた背景，3 章では本手法の詳細を述べる．第 4 章では実験，評価とそれに伴う考察，第 5 章に本研究のまとめ，今後の課題や展望を述べる．

2 従来のデザインパターン検出手法と問題

代表的な従来の検出手法の多くに静的解析を用いたパターン検出がある [3] [4] [5]．これらの手法は主にクラス構造の解析によって，条件記述に合致することでパターンが検出される．厳格な条件記述を用いてパターンを検出するため，あるクラスに複数のパターンのロールが重なって担わされている場合や，パターン適用の多様性に対して，検出の条件記述から少しでも外れた場合に検出の見落としが発生する．

パターンについての関連や継承，メソッド呼び出しなどをグラフ構造の集合として定義し，検出対象のプログラムも同様にグラフ構造にすることで，グラフ間の類似度からパターンを検出する手法がある [3]．しかし，図 1，2 のように，State パターン，Strategy パターンはクラス構造が類似するため判別が困難である．我々は，メトリクスと機械学習によってクラス構造からは読み取れない判別要素を導出し，上記のような構造が類似するパターンについて正確に判別した上での検出に取り組む．

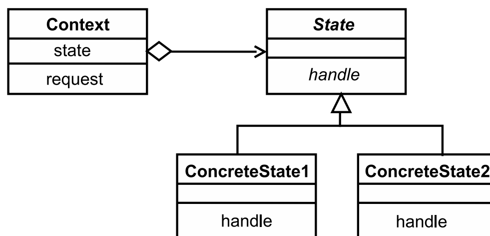

図 1 State パターン

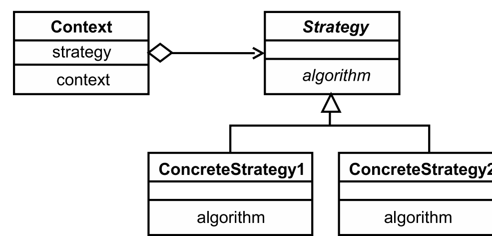

図 2 Strategy パターン

パターンごとのメトリクスによる測定値の特徴から，あるクラスが属しているパターンの候補を出力する手法がある [6]．しかし，この手法はおおまかにパターン候補を判別するため，最終的に手動で確認する必要があり属人性が高い．本手法では，メトリクスと機械学習による判別要素を用いるだけでなく，クラス構造も解析することで手動でフィルタリングすることなく検出する．

あらかじめ従来の静的解析手法によりパターンを検出した後，その結果を，パターンごとのクラスやメソッドに関する測定値情報を入力とした機械学習を用いてフィルタリングすることで精度を高める手法がある [7]．この手法は，事前に従来の静的解析手法を用いるため，前述した従来の検出手法の問題が発生する．本手法においても機械学習を要素技術として採用しているが，従来の検出手法を介することなくパターンを検出する．

パターン適用後におけるプログラムに加えて，適用前におけるプログラムを用いた検出手法がある [8]．この手法は，検出対象プログラムの改訂履歴を必要とする．本手法では，現時点の検出対象プログラムのみを解析しパターンを検出する．

パターンを検出しやすいように再分類し，クラスの相互間の構造や，システムの振る舞いなどからパターンを検出する手法がある [9] [10]．これらの手法では，パターンが重複して適用されている場合の検出が難しく，パターン適用の多様性による特殊な実装形態を考慮できていないという問題がある．本手法では，機械学習を用いることで，多岐にわたってロール候補を判定し，パターンの重複した適用や，

パターン適用の多様性に対応する．

パターンの検出には動的解析を用いた検出手法もある [11] [12]．これらは対象となるプログラムの実行経路などの情報からパターンを判別する．しかし，実行経路を網羅的に解析することが困難であり，断片的なクラス集合からの解析が難しい．

我々は上記に挙げたクラス構造が類似するパターンの判別や検出の見落としの問題に取り組む．

3 メトリクスと機械学習によるパターン検出手法

本手法は，定義学習フェーズ，検出フェーズの 2 フェーズから構成される．図 3 のように，定義学習フェーズに 3 プロセス，検出フェーズに 2 プロセスが含まれる．以下に，関係者として想定されるパターン専門家やプログラム保守者を含め，各プロセスを説明する．なお，本手法は対象となるプログラム言語として Java を扱う．

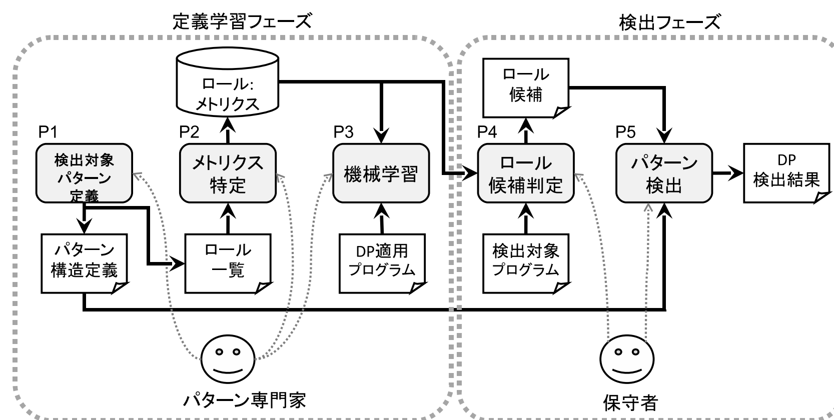

図 3　全体像

【定義学習フェーズ】

- P1：検出対象パターン定義
  パターン専門家は，検出対象とするパターンを特定する．そして，検出対象としたパターンの構造やパターンを構成するロールを定義する．
- P2：メトリクス特定
  パターン専門家は，メトリクスの特定手法である Goal Question Metric [13] により，P1 で定義したロールの判定に有用なメトリクスを特定する．
- P3：機械学習
  パターン専門家は，既にパターンが適用されたプログラムから，P2 で定義したメトリクスを用いてロール毎に測定値を取得し，機械学習システムに学習させる．学習後にロール候補判定の妥当性を検証し，検証結果が良好でなければ P2 に戻り，メトリクスを再考する．

【検出フェーズ】

- P4：ロール候補判定
  保守者は，検出対象のプログラムからクラス毎に，P2 で定義したメトリクスを用いて測定値を取得する．これらを機械学習システムに入力し，ロール候補を判定する．
- P5：パターン検出
  保守者は，P4 で判定されたロール候補，P1 で定義されたパターン構造定義（3.2 節の計算式 P，R，E に相当）を用いて，パターンを検出する．

3.1　定義学習フェーズ

P1：検出対象パターン定義

検出対象のパターンの構造や，パターンを構成するロールを定義する．現時点ではGoFパターンの中から，5パターン（Singleton，TemplateMethod，Adapter，State，Strategy）をとりあげ検出対象とする．GoFのパターンは生成，構造，振る舞いに関するパターンに分類でき，本手法では前述の5パターンによってこれらを網羅すると共に，クラス構造が類似するパターンの検出も試みる．現時点で扱う5パターンには計12ロールが含まれる．

P2：メトリクス特定

本手法では，メトリクスの決定手法であるGoal Question Metric [13]（以下，GQM）を用いて，検出対象の個々のロール候補判定に有用なメトリクスを特定する．GQMはメトリクスをトップダウン形式で決定するアプローチであり，トップダウンにメトリクスを特定することで目標とメトリクスの対応関係が明確になる．

我々は予備実験として，GQMを適用するのではなく汎用的なメトリクスの測定値を利用してロール候補判定の実験をした．しかし，規則性のない測定値をもたらすメトリクスの混在から，機械学習の学習結果が妥当なものではなかったため，GQMの採用に至った．Goal（目標）にロールの判定，Question（質問）に判定に到達するために答えなければならない質問，Metric（メトリック）に質問の答えに役立つメトリクスを特定する．また，本手法において，ロールが持つ属性や操作に着目してQuestionを特定することをメトリクス特定の指針とする．

例として図4においてGoalをTemplateMethodパターンのAbstractClassロールの判定と定める．そして，AbstractClassロールは抽象メソッドと，抽象メソッドを用いたロジックが書かれたメソッドを持つことから，これらのメソッドが定義されているかQuestionを定める．上記のAbstractClassロールの特徴から，メソッド数は抽象メソッド数より少なくとも1以上大きい値になるため，メソッド数と抽象メソッド数の割合から判別できると考える．よって，判別要素となるメトリクスとしてNOAM（Number of Abstract Methods）やNOM（Number of Methods）を特定した．

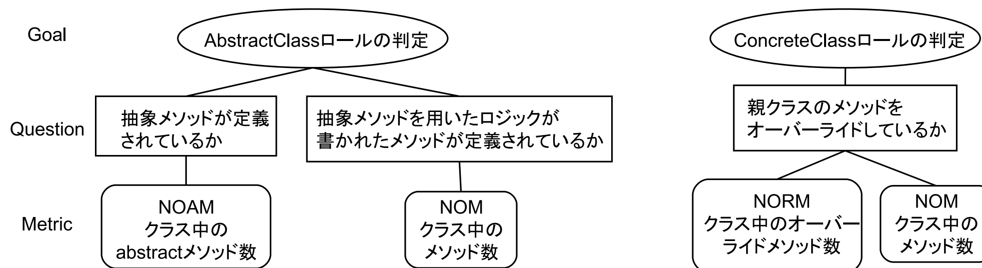

図4　GQMの適用例（TempateMethodパターン）

ロール候補判定の妥当性を検証した結果が良好でなければ，メトリクス特定→測定・学習→妥当性検証→メトリクス特定→ ··· といったフィードバックループを行い改善を図る．また，検出対象のパターンを追加する際には，追加したパターンを構成するロールにGQMを適用することで対応できる．

P3：機械学習

機械学習とは，コンピュータがサンプルデータを解析し，そこから未知のデータに対する予測のための有用な規則を取得する技術，手法である．

機械学習のアルゴリズムとしてニューラルネットワーク [14] を採用した．その他の機械学習アルゴリズムの1つに，線形入力素子を利用して2クラスのパターン識別器を構成するサポートベクターマシン [14] がある．しかし，本手法では，入力と

なるメトリクスによる測定値の依存関係を考慮し，多岐にわたるロール候補を判定したいため，ニューラルネットワークを採用した．多岐にわたる出力を実現することで，ひとつのクラスが複数のロールを担っている場合にも対応できる．

ニューラルネットワークは図 5 のように入力層，中間層，出力層から構成され，各層はユニットと呼ばれる要素から構成される．各ユニットからユニットへ入力される際に結合加重と呼ばれる重みが負荷され，結合加重の変化によって学習データから判定規則を取得する．結合荷重の調整方法の代表的なものがバックプロパゲーション（誤差逆伝播法）である．バックプロパゲーションは，図 6 のように得られた出力結果 y と正しい答え T との 誤差を計算し，結合荷重を出力に近い層から順に入力層まで調整し，ネットワークの出力誤差が基準を満たすまで調整していく．

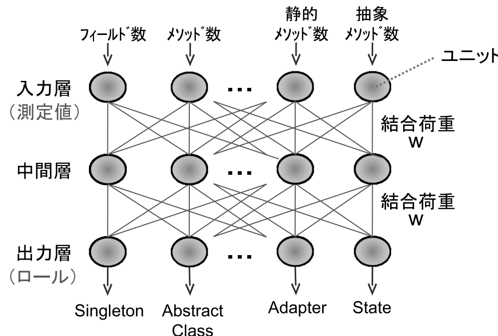

図 5　ニューラルネットワーク

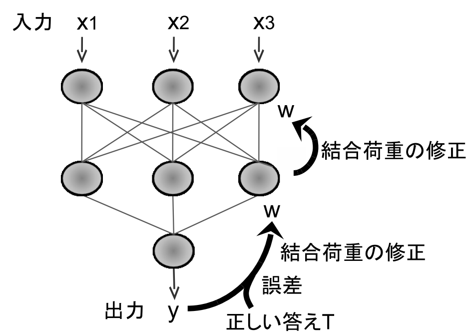

図 6　バックプロパゲーション

本手法では既存の階層型ニューラルネットワークシミュレータ [15] を利用する．本手法では学習法をバックプロパゲーション，ニューラルネットワークの階層数を 3 階層，入力層と中間層のユニット数を採用メトリクス数，出力層を判定対象ロール数としてニューラルネットワークを構成した．メトリクスを用いて，既にパターンが適用されたプログラムからロール毎に測定値を取得し，これらの測定値を入力，期待されるロールと出力として学習させる．学習回数の決定方法として，ニューラルネットワークシミュレータの機能である誤差曲線が収束する際の回数を学習回数とする．また，誤差曲線の収束判定は現時点では人手で判定することとする．

3.2　検出フェーズ

P4：ロール候補判定

検出対象のプログラムからメトリクスによる測定値を取得し，測定値を学習済みの機械学習システムに入力することで，判定対象のロール全てに対して 0-1 の間で出力される．そして，それらの和が 1 になるように正規化した値をロール適合度とする．ロール適合度が高いほど，明確に判定されたといえる．最終的に，検出対象ロール数の逆数を閾値として設け，閾値より高いものをロール候補とみなす．現時点では 12 ロールを扱うため，1/12=0.0834 を閾値とする．

例として，AbstractClass を担うクラスのサンプルコードからのメトリクス測定値は NOM（メソッド数）が 3，NOAM（抽象メソッド数）が 2 を示し，それ以外のメトリクスでは全て 0 を示した．この測定値を入力した際のロール候補判定結果を示した図 7 から AbstractClass しての出力値が最も高い値を示していることがわかる．この例では，図 7 の値を正規化することでロール候補として AbstratClass, Target として判定された．

P5：パターン検出

検出対象パターンの構造定義から，ロール候補間の関連を探索することでパター

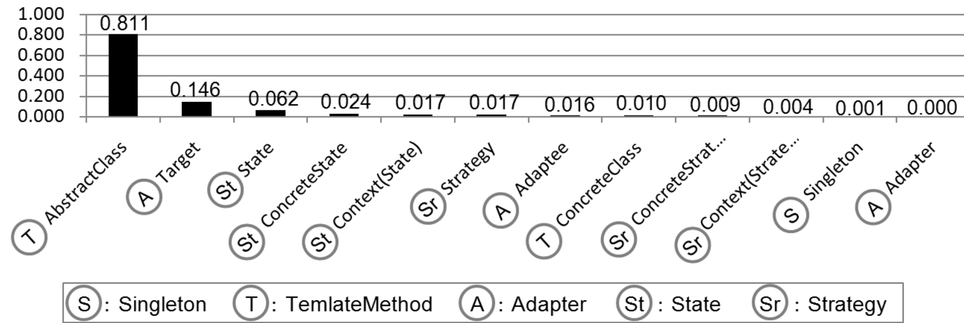

図 7　機械学習出力例グラフ

ンを検出する．本手法では，ロール候補の適合度が高いものから基点として探索を始め，パターンと構造的に適合すべきロール候補の全ての組み合わせに対して網羅的に探索する．ロール候補判定において検出されたロール候補を効果的に利用するために，検出対象パターンの構造定義の関連と方向が一致し，かつその関連の両端にあるロールとロール候補判定の結果が適合した場合にパターンとして検出する．また，パターン検出の際に関連の種類にも対応するために関連適合度を設ける．

現時点で扱う関連の種類として，継承，インタフェースの実装，集約の 3 つの関連を考慮する．既知のパターンが持つ関連と種類が一致する際は関連適合度を 1.0 とし，不一致の際は 0.5 とする．関連の種類が不一致の際の関連適合度を 0 とすると，以下に記述するパターン適合度も 0 となる場合があり，結果的にパターンとして検出されなくなる．これは静的構造を厳格に利用することになるため，従来の静的解析手法と同様の問題が発生する．関連適合度の値に関しては，検出結果に伴い最適な値を調整していく必要がある．また，今後の検出対象となるパターンの追加に伴い，考慮する関連の種類も検出対象パターンの定義のプロセスで追加していく．

本手法ではロール適合度と関連適合度を用いて，既知のパターンとの類似性を示すパターン適合度を算出する．既知のパターンについて，パターンを P，パターンを構成するロール集合を R，関連集合を E とする．また，対象プログラムについて，プログラム（クラス集合）を P'，ロール候補判定において最も高いロール適合度を示したクラス（探索基点）とし，P の構造に対応する E の要素と方向が一致してつながりがあり，かつ P の構造に対応する R の要素として判定されたプログラム中のクラス集合を R'，R' の要素間に存在する関連集合を E' とする．このとき，ロール適合度を $Role$，関連適合度を Rel，ある関連の両端にある 2 つのロールにおける $Role$ の平均値と Rel の積を Com，Com の平均値としてパターン適合度を Pat とし，以下の数式として定義する．我々は既知のパターンとの関連の相違による影響を，関連の両端の $Role$ に与えるために Com を定義した．また，Pat を $Role$ と同様に 0 以上 1 以下の値に正規化するために，Com の算出の際に 2 つの $Role$ の平均値をとり，Pat の算出の際にも Com の平均値を利用した．なお，以下に記述するパターン適合度の式表現のため関連の方向が不一致の Rel を 0 とする．

$$P=(R,E) \qquad P'=(R',E')$$
$$R=\{r_1,r_2,\cdots,r_i\} \qquad R'=\{r'_1,r'_2,\cdots,r'_k\}$$
$$E=\{e_1,e_2,\cdots,e_j\}\subseteq R\times R \qquad E'=\{e'_1,e'_2,\cdots,e'_l\}\subseteq R'\times R'$$

$Role(r_m,r'_n)$=学習済みの機械学習システムからの出力値　　ただし $r_m\in R$，$r'_n\in R'$

$Rel(e_p,e'_q)$=表 1 でロール間毎に定めた値　　ただし $e_p\in E$，$e'_q\in E'$

$$Com(e_p,e'_q) = \frac{Role(r_a,r'_b)+Role(r_c,r'_d)}{2} \times Rel(e_p,e'_q) \quad \text{ただし } r_a,r_c \in R, r'_b,r'_d \in R', e_p=(r_a,r_c), e'_p=(r'_b,r'_d)$$

$$Pat(P,P') = \frac{1}{|\{(e_p,e'_q) \in E \times E' | Rel(e_p,e'_q)>0\}|} \sum_{e_p \in E, e'_q \in E'} Com(e_p,e'_q)$$

図 8 に TemplateMethod パターンの検出例を示す．この例では，クラス SampleA が AbstractClass として最も高いロール適合度を示したとして上記のパターン適合度の算出アルゴリズムに則り，図 8 の検出対象のプログラム Samples と TemplateMethod パターンのパターン適合度を算出する．算出過程は以下のとおりである．

$TemplateMethod=(R,E)$ $\qquad Samples=(R',E')$

$R=\{AbstractClass,ConcreteClass\}$ $\qquad R'=\{SampleA,SampleB,SampleC\}$

$E=\{AbstractClass \triangleleft\!- ConcreteClass\}$ $\qquad E'=\{SampleA \triangleleft\!- SampleB, SampleA \leftarrow\!\diamond SampleC\}$

$Role(r_1,r'_1)=0.82 \qquad Role(r_2,r'_2)=0.45 \qquad Role(r_2,r'_3)=0.57$

$Rel(e_1,e'_1)=1.0 \qquad Rel(e_1,e'_2)=0.5$

$Com(e_1,e'_1)=\frac{(0.82+0.45)}{2} \times 1.0=0.635 \qquad Com(e_1,e'_2)=\frac{(0.82+0.57)}{2} \times 0.5=0.348$

$Pat(P,P')=\frac{1}{2} \times (0.635+0.3475)=0.492$

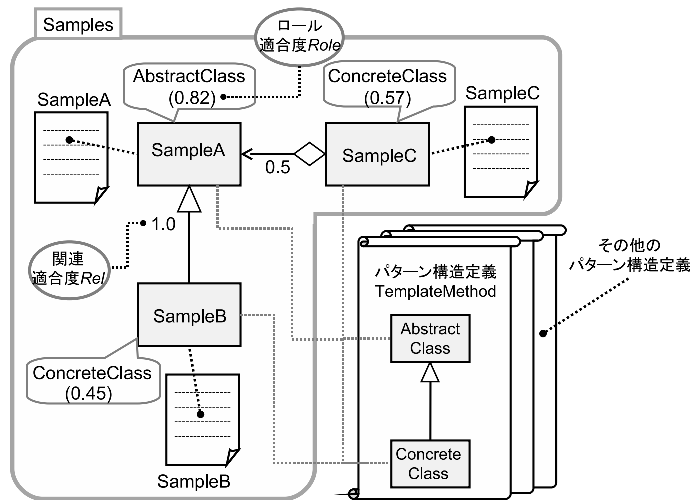

図 8　パターン検出例（TemplateMethod パターンの場合）

上記より，図 8 のプログラムは TemplateMethod パターンのパターン適合度 0.492 として算出された．パターン適合度はロール適合度と同様に 0 以上 1 以下の値に正規化された値であるため，本手法では，ロール候補判定と同様に検出対象ロール数の逆数（0.0834）を閾値として，パターン適合度が閾値より高いものを最終的な検出結果として出力する．図 8 のプログラムはパターン適合度 0.492 ＞ 0.0834 であるため，TemplateMethod パターンとして検出されたとみなす．

4　評価および考察

我々は，機械学習システムに学習させた結果，ロールの判別要素を導出できているかを確認するためにシステムの妥当性を検証した．また，本手法の検出精度や再

現率を検証し，本手法が構造が類似するパターンの検出やパターン適用の多様性に対応できているか確認するために従来の検出手法との比較実験を行った．

4.1 ロール候補判定の妥当性検証

ロール候補判定の妥当性の検証方法として，交差検証（Cross Validation）を用いた．これはデータを n 個の群に分割し，そのうち 1 個のデータ群をテストデータ，残りの n-1 個のデータ群を学習データとしてテスト実行しシステムの妥当性を検証する手法である．我々はデータ数を 5 分割しテスト実行を 5 回行った．データとしてパターンサンプルコード（計 50 箇所）[1] [16] [17] と，実コード（Java ライブラリ [18]，JUnit [19]，Spring Framework [20] から計 158 箇所）のパターン適用箇所を用いた．実コードのパターン適用箇所に関しては人手により定性的に収集した．

サンプルコードを用いた場合と，実コードを用いた場合の採用メトリクスを表 1 に示す．サンプルコードを用いた場合と実コードを用いた場合の採用メトリクスの相違点として，サンプルコードでは，State パターンの ConcreteState ロールの状態遷移のメソッドで他の ConcreteState ロールを生成する点から，NMGI（Number of Methods Generating Instance）を採用した．実コードでは，パターンの構成に不要な属性，操作も含まれて実装され，上記のような State パターンに特化した実装が少ないため，NMGI を除外した．現時点で第三者が本手法を利用する場合，検出対象プログラムが [16] [17] のような明らかにパターンのサンプルコードに類似する際は NMGI を採用し，実コードに類似する場合は NMGI を採用しないこととする．

表 1　採用メトリクス（○：採用，×：不採用）

略称	内容	サンプル	実コード
NOF	フィールド数	○	○
NSF	静的フィールド数	○	○
NOM	メソッド数	○	○
NSM	静的メソッド数	○	○
NOI	インタフェース数	○	○
NOAM	抽象メソッド数	○	○
NORM	オーバーライドしているメソッド数	○	○
NOPC	private コンストラクタ数	○	○
NOTC	オブジェクト型の引数を持つコンストラクタ数	○	○
NOOF	オブジェクト型のフィールド数	○	○
NCOF	自身の型のフィールドを持つ他クラス数	○	○
NMGI	インスタンスを生成している処理を行うメソッド数	○	×

本手法はパターン適用の多様性に対しての網羅的な検出を目的としているため，本節では再現率のみを議論することとする．再現率とは検出結果にどれだけ漏れがないかを示しており，それに対して精度は検出結果にどれだけ非適合検出がないかを示している．再現率は表 2 の交差行列をもとに以下の計算式より算出される．表 2 の行は検出対象に対して適合しているかどうかを示し，列は検出されたかどうかを示している．なお，w_r，x_r，y_r，z_r はロール数を示し，w_p，x_p，y_p，z_p はパターン数を示している．

$$\text{ロール候補判定の再現率：} Re_r = \frac{w_r}{w_r + x_r}$$

[1] 全サンプルコード取得先：http://www.washi.cs.waseda.ac.jp/ja/paper/uchiyama/fose09.html

表 2　交差行列（ロール数，パターン数）

	検出された数	検出されなかった数
適合数	w_r, w_p	x_r, x_p
非適合数	y_r, y_p	z_r, y_p

上記の 5 回のテスト実行において，実際にデザインパターンが適用されているロールごとの再現率の平均値を表 3 に示す．State，Strategy パターンは，同じクラス構造を持つためロール候補判定時で正確な判別が必要になる．したがって，State，Strategy パターン以外はロール適合度が閾値以上の場合に正確に判断できたとみなし，State，Strategy パターンに関しては閾値以上かつ両者の判別ができた場合に正確に判断できたとみなした．

表 3 の考察として，実コードを用いた場合がサンプルコードを用いた場合と比べて全体的に低い再現率を示した．実コードには，パターンの構成に不要な属性，操作が含まれているため，正確な判定が困難であったと考えられる．したがって，様々な実装形態を網羅するためにより多くの学習データを収集することが必要と考えられる．

実コードを用いた場合に関して State パターンに着目したとき，Context ロールの検出は高い再現率となったが，State ロールや ConcreteState ロールは特に低い再現率を示した．しかし，閾値以上の場合のみを考慮した場合，State ロールと Strategy ロールは判別できなかったが，高精度で State ロール候補として出力された．したがって，パターン検出部により Context を基点として探索をはじめることで State パターンとして判別され最終的な検出精度は向上すると考えられる．

表 3　ロール候補判定の再現率

		再現率の平均値（%）	
パターン名	ロール名	サンプル	実コード
Singleton	Singleton	100.0	84.7
Template Method	AbstractClass	100.0	88.6
	ConcreteClass	100.0	58.5
Adapter	Target	90.0	75.2
	Adapter	100.0	66.7
	Adaptee	90.0	60.9
State	Context(State)	60.0	70.0
	State	60.0	46.7
	ConcreteState	82.0	46.6
Strategy	Context(Strategy)	80.0	55.3
	Strategy	100.0	76.7
	ConcreteStrategy	100.0	72.4

4.2　パターン検出結果

サンプルコード，実コードのそれぞれにおいて，パターン適用箇所をテストデータとして本手法でパターンを検出し評価する．表 4 に本手法のパターン検出における精度と再現率を示す．再現率や精度は表 2 をもとに以下の式で算出される．

パターン検出の再現率：$Re_p = \frac{w_p}{w_p + x_p}$　　パターン検出の精度：$Pr_p = \frac{w_p}{w_p + y_p}$

テストデータとしてサンプルコード，実コードを用いた場合の共通している点として，両者とも全体的に精度より再現率が高い値を示した．このことから本手法の検出の見落としを防ぐ目的と合致しているといえる．しかし，特に実コードにおける精度の値から State，Strategy パターンの非適合検出が多くみられたことがわかる．これに対してはパターン検出の際のより最適な閾値を検討する必要がある．

サンプルコードを用いた場合では，全体として 9 割以上の再現率を示したのに対して，実コードの場合は 6 割以上の再現率を示した．

実コードを用いた場合，サンプルコードを用いた場合と比べて特に Adater パターンの再現率が低い結果になった．これは，表 3 から Adapter パターンのロール候補判定の再現率が全体としてそれほど高くなかったことが原因と考えられる．最終的にパターンとして検出されるには，ロール候補判定の閾値を用いる際にパターンを構成する全てのロールが閾値より高いロール適合度を示している必要がある．Adapter パターンの場合，パターンを構成するロールのいずれがロール候補として判定されなかったクラスが多くあり，結果的にパターン検出の再現率が低い値を示した．これに対する解決策として，GQM によるメトリクス特定のプロセスに戻り，メトリクスを再考する必要性がある．

実コードにおける State パターンの検出に関しては，4.1 節で述べたように Context を基点として探索をはじめることで State パターンとして判別され，ロール候補判定の再現率と比べて高い値を示したと考えられる．

表 4　パターン検出の精度と再現率

	テストデータ数		精度（%）		再現率（%）	
パターン名	サンプル	実コード	サンプル	実コード	サンプル	実コード
Singleton	6	6	60.0	63.6	100.0	100.0
Template	6	7	85.7	71.4	100.0	83.3
Adapter	4	7	100.0	100.0	90.0	60.0
State	2	6	50.0	40.0	100.0	66.6
Strategy	2	6	66.7	30.8	100.0	80.0

4.3　従来の検出手法との比較実験と考察

我々は容易に実装を入手できる従来の検出手法 [3] との比較実験を行った．サンプルコード，実コードのそれぞれにおいて，パターン適用箇所をテストデータとして両手法でパターンを検出し評価する．また，本手法の機械学習の学習データにも，実コードのパターン適用箇所から，実験データとは異なるものを用いた．

図 9 に両手法の検出結果の再現率-精度グラフを示す．これは検出結果から再現率と精度を算出しグラフ化したものである．再現率-精度グラフによって 2 つのシステムを評価する場合，どちらの曲線が上にあるかで判断でき，上にあるシステムのほうが性能が高いと評価できる．再現率や精度は検出結果を順位付けすることで算出した．従来の検出手法ではパターン検出結果に順位付けするための数値がないため，検出結果の中で適合，非適合のものを交互に並べて算出した．

図 9 から両手法においてサンプルプログラムを用いた場合のほうが高精度な結果を得た．これは，サンプルコードには実コードのようにパターンの構成に不要な属性，操作が含まれていないため，判別しやすかったと考えられる．

表 5 に実コードを用いた実験で判別できた State，Strategy パターンにおける Con-

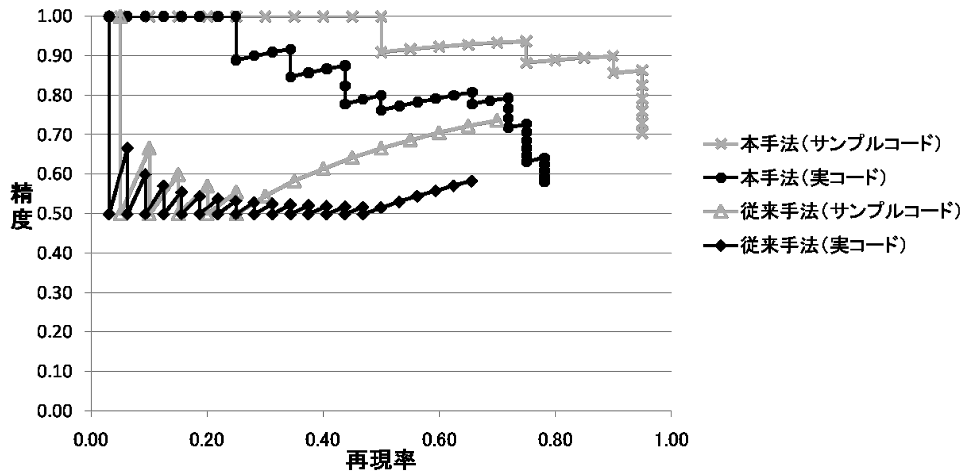

図 9　再現率-精度グラフ

text ロールのメトリクス測定値の抜粋を示す．State パターンは State ロールで状態を扱い，状態におけるアクションに関わる実装を Context ロールで行う．Strategy パターンは各アルゴリズムの処理を Strategy ロールにカプセル化することで，Context 内での処理が State パターンに比べてシンプルになる．表 5 より，State パターンの Context ロールでフィールド数，メソッド数共に最も高い場合はそれぞれ 45，204 であり，Strategy パターンの Context ロールでは，それぞれ 18，31 であったことから，上記のような両パターンの Context ロールの複雑さがフィールド数やメソッド数に現れ，判別要素になったと考えられる．従来の検出手法では，State，Strategy パターン以外のパターンは高精度に検出できたが，State，Strategy パターンを判別できないため，図 9 のような結果を得た．

表 5　Context ロールにおけるメトリクスの測定値例

パターン-ロール名	フィールド数	メソッド数
State-Context	12	58
	45	204
	11	72
Srategy-Context	18	31
	3	16
	3	5

本手法では，既知のパターンの形に近いものも検出できた．しかし，本手法はメトリクスと機械学習を利用することで，厳格な条件記述を用いた従来の検出手法と比べて緩やかな検出になるために False Positive が発生した．特に Singleton パターンの FalsePositive が目立った．Singleton パターンのような 1 ロールから構成されるパターンの判定は，ロール候補判定で検出結果が決まるため，そのロールに特化したメトリクスを再考する必要がある．

上記を踏まえ総合的に評価すると，図 9 より本手法の曲線が従来の検出手法より上に位置するため従来の検出手法に比べて性能が優れていると判断できる．

5 おわりに

我々は，メトリクスの測定値を入力とした機械学習を用いてロール候補を判定し，クラスの関連情報からパターンを検出する手法を提案した．そして，クラス構造が類似するパターンの判別や検出の見落としの問題に取り組んだ．

従来の検出手法との比較実験結果から，本手法ではクラス構造が類似するパターンを判別することで従来の検出手法と比べて高精度な検出を確認できた．また，既知のパターンに近い形のものも検出したことから，広範囲な検出を実現できた．しかし，従来の検出手法に比べて緩やかな検出のために False Positive が発生した．

今後の展望として以下の内容が挙げられる．

- パターン適用の多様性を網羅するようなより多くの学習データ収集
- 結果がデータに依存している可能性も考えられるため，GQM 適用工程にフィードバックしメトリクスの再考によってロールに特化したメトリクスを導出
- False Positive の抑止や，検出対象のパターンの追加
- 各適合度などのパラメータや式に関する正当性，妥当性の検討
- 機械学習の誤差曲線における，機械的に属人性を排した収束判定による学習回数の決定，学習回数と精度の相関関係の検討

参考文献

[1] E.Gamma, R.Helm, R.Johnson, and J.Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[2] M.Lorenz, J.Kidd *Object-oriented Software Metrics*. Prentice Hall, 1994.

[3] N.Tsantalis, A.Chatzigeorgiou, G.Stephanides, and S.Halkidis. Design Pattern Detection Using Similarity Scoring. *IEEE Trans. Software Engineering*, Vol.32, No.11, 2006.

[4] A.Blewitt, A.Bundy, and L.Stark. Automatic Verification of Design Patterns in Java. In *Proceedings of the 20th International Conference on Automated Software Engineering*, 2005.

[5] 金子崇之, 深澤良彰. プログラムからのデザインパターン抽出技法とその評価. *Technical Report of IEICE*, SS98-40. 1999-01.

[6] H.Kim and C.Boldyreff. A Method to Recover Design Patterns Using Software Product Metrics. In *Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability*, 2000.

[7] R.Ferenc, A.Beszedes, L.Fulop and J.Lele. Design Pattern Mining Enhanced by Machine Learning. *21st IEEE International Conference on Software Maintenance*, 2005.

[8] H.Washizaki, K.Fukaya, A.Kubo, Y.Fukazawa. Detecting Design Patterns Using Source Code of Before Applying Design Patterns. *8th IEEE/ACIS International Conference on Computer and Information Science*, 2009.

[9] N.Shi and R.A.Olsson. Reverse Engineering of Design Patterns from Java Source Code. *21st IEEE/ACM International Conference on Automated Software Engineering*, 2006.

[10] H.Lee, H.Youn and E.Lee. Automatic Detection of Design Pattern for Reverse Engineering. *5th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2007)*, 2007.

[11] L.Wendehals and A.Orso. Recognizing Behavioral Patterns at Runtime Using Finite Automata. In *4th ICSE 2006 Workshop on Dynamic Analysis*, 2006.

[12] S.Hayashi, J.Katada, R.Sakamoto, T.Kobayashi, M.Saeki. Design Pattern Detection by Using Meta Patterns. *IEICE Transactions, Vol.91-D No.4*, pp.933-944, 2008.

[13] V.R.Basili and D.M.Weiss. A Methodology for Collecting Valid Software Engineering Data, *IEEE Transactions on Software Engineering, Vol.10, No.6*, 1984.

[14] T.Segaran. 集合値プログラミング. オライリージャパン 2008.

[15] 平野廣美. C++と Java でつくるニューラルネットワーク. パーソナルメディア. 2008.

[16] 結城浩. Java 言語で学ぶデザインパターン入門. SoftBank Creative. 2004.

[17] 田中宏和. Java で Hello World!. *http://www.hellohiro.com/pattern/*

[18] Sun Microsystems. Sun Developer Network. *http://developers.sun.com/*

[19] JUnit.org. Resources for Test Driven Development. *http://www.junit.org/*

[20] SpringSource.org. Spring Source. *http://www.springsource.org/*