自然言語処理技術によるテスト・ケース文書の自動分類手法

海野 裕也　中村 大賀

Automatic Classification of Test Case Documents Using Natural Language Processing

Yuya Unno and Taiga Nakamura

テスト・ケースを分類・整理することでテストの品質や効率を評価し最適化する手法が近年注目されているが，その分類作業はすべて人手で行われているため，作業コストが問題となっている．筆者らは，自然言語処理技術を応用してテスト記述のテキストを解析し，ODC (Orthogonal Defect Classification) に基づく分類作業を自動化する手法を新たに開発した．本手法は実際のテスト・ケース文書を観察して得た定性的な特徴を反映させており，特にテスト・ケースの類似性を利用してテスト記述の中から分類に影響しないテスト準備の部分を取り除き，その上で分類に特徴的な単語列パターンを抜き出すことで動作する．人間が分類した結果との一致率で自動分類の性能を評価した結果，90% 近い高い適合率と再現率を示した．これにより従来人手のみで行われていたテスト・ケースの分類作業が飛躍的に効率化された．

Classifying test case documents is an important task in recent approaches for quality improvement and optimization of testing activities. However, manual classification tends to be too expensive. We present a new method for automatic classification of test cases using ODC (orthogonal defect classification) by applying natural language processing technology to test case texts. The key ideas of our method reflect insights from the observation of the characteristics of actual test case documents. In particular, it detects and excludes preparation steps to eliminate noise, and then classifies the rest of the steps by identifying word sequence patterns by characterizing classification types. We evaluated the performance of the method by comparing the output with manual classification results, and confirmed approximately 90 percent correctness. The method can greatly reduce the amount of manual classification tasks.

Key Words & Phrases：テスト・ケース分類，ODC，自然言語処理，Total Test Quality
Test case classification, ODC, natural language processing, Total Test Quality

1. 背景

1.1 Orthogonal Defect Classification

ソフトウェアの規模と複雑性が増すにつれて，テスト工程の重要性は上がっている．テストに費やせる時間や人的資源は限られている上，ソフトウェアの欠陥を見つけられずにリリースすると莫大な損失に結びつくため，効率的で網羅的な品質の高いテスト工程が求められている．

テストの効率を評価する手法として，IBM で開発され広く普及しているものの 1 つが Orthogonal Defect Classification（以下，ODC）である［2］［4］．ODC では，テストで検出される欠陥を欠陥の報告担当者・対応担当者ごとに複数の観点軸で分類し，それらを元に欠陥の傾向を分析する．テスト・ケースの分析に ODC を適用する場合には，各テスト・ケースによって検出できる欠陥種別に基づいてテスト・ケースを分類する．多数のプロジェクトから同じ基準で，この分類結果を収集・蓄積することにより標準的な分類結果の分布を知ることができ，この標準的な分布と比較することで，テストの適正さを評価することができる．テストが手薄であることが分かればその部分のテストを強化すべきである．また，逆に無駄なテスト・ケースを省略することによって最適化が図れる．

こうした手法は，社内のプロジェクトにおけるテストの効率化を図るだけではなく，お客様の既存テスト・ケースを分析して提言を行うこともできるとみられている．さらに，ODC の手法は学術的にも評価されており，例えば Butcher ら［1］が現実のプロジェクトにおけるテスト工程に対して ODC を用いて評価し，テスト効率を効果的に向上可能であることをケーススタディーにより示した例などが挙げられる．

しかしながら，従来テスト・ケースを分類する作業は基本的にすべて人手で行わなければならなかった．そのため，対象とするプロジェクトの規模が大きいとテスト・ケースの ODC 分類の作業コストが肥大化する問題があった．本稿は，こうした背景を踏まえてテスト・ケースの ODC 分類を自動化す

提出日：2009年9月7日　再提出日：2010年3月23日

る新しい手法を提供する．既存テスト・ケースを自動的に分類することができれば上記の手順における最大のボトルネックが解消され大きな価値がある．自動分類された結果をそのまま使わずとも，人手で分類する際のヒントとして用いることによって作業を効率化できる．

1.2 テキスト解析技術

本稿が対象とするテスト・ケースは，いわゆるユニット・テスト以降のテストで，システムの操作手順と確認内容で構成されるものである．多くのプロジェクトでは，こうしたテスト･ケースはテキストで記述されている．テスト・ケース自体がプログラムであるユニット・テストと異なり，このような非定型の自然言語で書かれたテスト・ケースの分類技術は確立されていない．本稿では，自然言語の解析に有用である IBM のテキスト解析技術，LanguageWare［6］をベースとした手法を構築する．本稿で用いたデータは英語で記述されたテスト･ケースであるが，LanguageWare は多くの主要言語に対応しており，多言語版への拡張は比較的容易に行えることが期待される．

2. 問題の定式化

2.1 テスト・ケース文書のモデル化

実プロジェクトで作成されるテスト・ケースのファイル形式やフォーマットはプロジェクトによってさまざまである．こうした多様な形式を統一的に扱うために，本稿ではテスト・ケース文書を図 1 のようにテスト・セット，テスト・ケース，ステップの 3 段階構造でモデル化する．

テスト・ケースを記述する情報の最小単位がステップであり，ステップの説明の内容が分類のための最小粒度の情報となる．ステップには「ログインする」などの操作を示す実行ステップと，「正しくログインできる」などの確認・検証を示す検証ステップがある．1 つ以上のステップ列を一連の操作のまとまりとしてまとめたものがテスト・ケースである．一般に，テスト時にはテスト・ケース単位で実行の日時や実行者，結果などが管理される．さらに，同一のプロジェクトや 1 つのアプリケーションに対するテストなど，関連するテスト・ケースをテスト・セットとしてまとめる．この 3 段階構造は，Rational Quality Manager (RQM) v1［7］で "manual test case" を定義する場合のスキーマとも一致する．われわれのアプローチでは，現実のプロジェクトで作られる多様なテスト・ケース文書をこの 3 段階の構造に当てはめることにより，統一的な手法で分類を実行できるようにする．

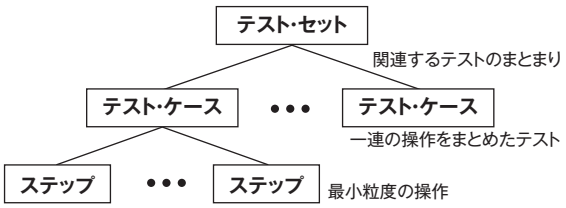

図 1. テスト・ケース文書のモデル

テスト・ケース文書のどの部分をどのレベルの情報として切り出すかはユーザーが選択できる．例えば，テスト・ケース文書が Excel ファイルで記述されているとする．シートごとにテスト･ケースが記述され，各シートの 1 行が各テストのステップにおける "test case action" および "expected results" の情報を含む．そこで，ファイル全体をテスト・セットに，シートをテスト・ケースに，シート内の "test case action" を実行ステップ，"expected results" を検証ステップとして抽出することでテスト・ケースが得られる．

テスト・ケース文書の構成によっては，異なった切り出し方が生じ得る．先の例では，テスト・セットをシートごとに分け，各シートの 1 行をテスト・ケースとすることも可能である．この場合，ステップ自体は同じだが，ステップからテスト・ケースへのくくり方が変わる．つまり，1 テスト・ケースが，実行，検証ステップ 1 つずつで構成される短いケースになる．われわれの分類手法は，ユーザーが選択した任意の切り出し方に対応して分類を実行することが可能である．

個々のテスト・ケースには，分類コードなど，上記の構造に当てはまらないプロジェクト固有の記述も含まれている．これらが分類に有用である可能性もあるが，今回それらの情報は使用しない．特殊な記述に頼らないことで，本手法が汎用かつ頑健に動作することが期待される．

2.2 テスト分類の定式化

ODC では，ソフトウェアの含んでいる欠陥が実際に表面化する環境や再現手順などの条件をトリガーと呼ぶ．例えば，境界条件の値を入力することで問題が生じる場合，「境界条件を入力する」ことがトリガーである．トリガー・タイプはこのようなトリガーをプロジェクト非依存な形で分類するものであり，上記の例は variation というタイプになる．具体的なトリガー･タイプの一覧は文献［3］を参照されたい．

本稿ではテスト・ケースに対するトリガー・タイプの分類を行う．1 つのテスト・ケースが複数ステップから構成されるため，各テスト・ケースの検証ステップのどこで問題が検出されたかによってトリガー･タイプが異なる可能性がある．この場合，ステップごとに求めたトリガーを統合してテスト・ケース全体として何のトリガーに当てはめるべきかを定める．

2.3 手動分類結果で得たテスト・ケースの特徴

各トリガーの定性的な特徴を理解するために，現実のプロジェクトで使われたテスト・ケースを手動で分類した．複数の分類者が独立，効率的に分類作業を行えるよう Web アプリケーションを作成した．なお，ここで分類した結果は性能評価実験のテストデータとしても利用される．

213 のテスト・ケース文書，2,114 のステップに関して人手で分類を行った．そこから得られた各トリガーに関する定性的な特徴を示す．なお，このテスト・ケース文書群の中に，下記の 4 つのトリガー・タイプ以外は出現しなかった．

1）Coverage: 1 つの機能の基本的なテスト．想定される入力に対して機能通りに動作することを確認する．入手したテスト・ケースはプロジェクトに関係なく大多数が coverage に

分類される．このことは，明示的にドキュメント化されたテスト・ケースが，正常ケースの機能確認に偏っていることを示す．

2）Variation: 異常ケースの入力に対する動作を確認するテスト．エラー・ケースの動作確認はほとんど variation に分類される．

3）Sequencing: 複数の機能を組み合わせて動作させた場合のテスト．ロック順序などステップの順序が重要となるテスト・ケースがこれに分類される．

4）Work/stress: リソースの上限や下限が要求どおりか調べるためのテスト．パフォーマンス・テストのためのテスト・ケースなどが分類される．

特に顕著なのが，トリガーの出現頻度の偏りである．特徴的な表現のないテスト・ケースは正常ケースの確認がほとんどで，coverage に分類される．またそれ以外もほとんどが variation に分類される．この 2 つだけで 95% 以上を占め，残りのトリガーに分類されるテスト・ケースはごくまれである．この傾向は Chillarege と Prasad の定量的な実験結果ともよく一致している［3］．

各ステップを観察すると，1 つのテスト・ケースで検査したい事象を直接検証しているステップは，ほとんどの場合ごく一部のステップに限られていた．例えば，**表 1** は太字で示したステップで不正な値を入力させる，variation のテスト・ケースである．このテスト・ケースの中で，本質的に検証したい事象は太字のステップだけに現れており，例えば最初の login などのステップはそのための準備に過ぎない．準備のためのステップは，トリガー・タイプに関係なくほとんどのテスト・ケースに出現するが，成功を前提とした操作を行うので，単体では coverage に分類されてしまう．従って，こうした準備ステップを正しく識別する必要がある．

準備ステップの特徴として，まったく同じステップが別テスト・ケースに出現することが多いことが分かった．例えば**表 2** は，**表 1** と同じテスト・セット中に含まれるテスト・ケースである．表中に太字で示した 2 つのステップは，**表 1** の 2 ステップと should のスペル・ミスも含めて字面が完全に一致している．ログインやユーザー名の入力といった準備ステップは，多くのテストで共通なので，コピー・ペースト機能を使用して作成されたためと考えられる．準備ステップがほかのテスト・ケースに出現する傾向は多くのプロジェクトに共通で確認された．そのため，コピー・ペーストを検出することで準備ステップと判断することは妥当だと考えられる．

手動でテスト・ケースを分類する際，それぞれの記述に含まれる特徴的な表現に注目することが有用であった．例えば，**表 1** 中で variation のチェックを行っている太字のステップの中には，異常値を入力したことを示唆する illegal という単語が見られる．これ以外の例では，異常値の結果としてアプリケーションの動きが阻害されたことを示唆する should not や would not といった単語列も頻繁に出現する．これらの単語はプロジェクトとは無関係に特徴的に出現する．

表1. テスト・ケース中のステップの例（プロジェクト固有名は匿名化済み）

ID	記述
TEST-10-1.1	Eagle Login & Open Project Login to Eagle with user who has Price access & Open a Project by selecting it from the Project Grid of Eagle Main Browser
TEST-10-1.1	Eagle Login shld be successful & the Project should be opened
TEST-10-1.2	Release Project Try to release the project after importing the product to Typesite & to Scenario
…	
TEST-10-1.6	**Validation of Grid Fields Enter some illegal values to the grid fields**
TEST-10-1.6	The validation part in the Grid will only check the length of the entered values. It will not check for illegal characters

表 2. 準備ステップの例（プロジェクト固有名は匿名化済み）

TEST-7-1.1	**Eagle Login & Open Project Login to Eagle with user who has Price access & Open a Project by selecting it from the Project Grid of Eagle Main Browser**
TEST-7-1.1	**Eagle Login shld be successful & the Project should be opened**
TEST-7-1.2	Users Defined Views In the Eagle toolbar select "Save View", give the name to be saved.
TEST-7-1.2	The users Defined view should be saved

表 3. トリガー・タイプと単語出現頻度比較

単語	出現ファイル数	Var/cov 比
missing	3	195.30
characters	4	83.70
invalid	6	48.82
error	7	48.82
spaces	3	41.85
were	3	34.87
But	5	34.87
validation	4	34.87
more	4	34.87

このことを定量的に評価するために，手動分類の結果を使い，単語出現頻度の偏りを調べ，件数の多い coverage と variation に分類されたステップに関して，各単語の出現割合を調べた．そして coverage と variation における出現割合の比をとり，上位 10 件を調べたのが**表 3** である．各単語 W に対して，Var/cov 比は，

$$\frac{(\text{Wを含むvariation数})/(\text{全variation数})}{(\text{Wを含むcoverage数})/(\text{全coverage数})}$$

で計算している．ただし，特定のプロジェクトで出現する単語に偏らせないために，3 つ以上のファイルで出現した単語に限っている．出現ファイル数も表中に記した．

未入力の異常ケースを示唆するmissingや，不正入力を示唆するinvalid，また異常結果を示唆するerrorなど，variationタイプの判断として汎用的に使えそうな単語を発見できる．一方で，それ以外の単語を見ると，例えばcharactersやspacesといった分類には使えなさそうな単語も多数出現している．しかし，これらの単語の周辺をよく観察すると，例えばcharactersは不正な文字入力の意味でinvalid charactersという形で，あるいは入力文字数の検査をするために15 charactersという形で出現している．また，spacesもスペース文字を例外として扱うためにdoes not contain spacesというふうに出現する．これらの単語の出現をそのまま分類の判断に使うのは誤分類を増やす危険があるが，これらの周辺から分類のための特徴を探すのは有用と考えられる．

3. テスト・ケースの自動分類手法

テスト・ケースを各トリガー・タイプに分類するために，われわれの手法は3段階の工程を経る．まず，準備ステップを検出し，分類対象から除外する．次に除外されなかったステップに対して，それぞれパターンによる分類を行う．最後に，ステップごとに行った分類結果を集約することで1つのテスト・ケースの分類結果とする．

3.1 準備ステップの検出

テスト・ケースの分類に直接関係しない準備ステップの記述は分類時にノイズとなるので，準備ステップを除去する．これを検出するために，コピー・ペーストの検出を行う．ステップの記述を単語列に変換して，編集距離※1の小さい記述は同一であるとみなす．同一とする編集距離の上限をdとする．重複単語列の検出にはRabin-Karpの文字列検索アルゴリズム［5］を用いた．このアルゴリズムは部分文字列の一致個所を高速に探索するものである．各部分文字列のハッシュ値を計算することで，直接文字列比較せずに比較を行える．単純な手法では平均単語列長の自乗に比例する時間がかかるが，このアルゴリズムは平均計算時間が単語列長に対して線形で行うことができる．

編集距離がd以下の単語列の検出も，このアルゴリズムの拡張で効率的に行うことができる．各単語に対して，削除と挿入の操作を合計d回行った単語列をすべて用意し，Rabin-Karpアルゴリズムを適用する．削除と挿入の組み合わせは，dに対して組み合わせ爆発を起こすが，dは1や2など非常に小さな値までしか許さないため問題にならない．特に，この場合も入力長に対して線形時間で実行できることに注目されたい．

※1　編集距離（edit distance）とは，2つの文字列の差異の大きさを表す数値であり，一方の文字列をもう一方に変形するのに必要な挿入，削除，置換の最小回数で表される．

3.2 ステップごとの分類

各ステップの分類は独立に行う．まず，各トリガー・タイプに頻出する単語列パターンを用意する．ここで用意するパターンは，主に2.3節で予備調査された頻出単語を元に設計される．例えば,variationのパターンとして「error」といった単語単体から「would not+ 動詞」といった複合単語の連続までさまざまである．

実際に使用するパターンは，機械的に作るのではなく，一般性を損なわないよう人手で選択した．こうしたことには2つ理由がある．1つは，プロジェクトごとに分類に寄与しそうなパターンが大きく異なるため，単純に高頻度パターンを利用すると特定のプロジェクトだけに最適な分類器ができる可能性があったため．もう1つは，正解例が少なかったため，用意したデータに対して過剰に特化する恐れがあったためである．

分類対象のテストデータが与えられたら，まず記述内容を単語に分割し，各単語の品詞を決定する．この処理はLanguageWareを用いる．そして，分解された単語列に対して，上記のパターンの出現頻度を数え，パターンの出現頻度に基づいて各ステップを分類する．ここでは，最も頻度の高いパターンを持っていたトリガー・タイプにステップを分類する．なお，具体的な単語分割とパターンの仕様は付録Aに記した．

なお，英語のように空白文字で単語が分割された言語では単なる単語分割だけなら文字情報のみで行える．ただし，この場合原型や品詞を使えないため，errorとerrorsを区別して記述する必要があり，またwill+ 動詞のように品詞でパターンを指定することができない．さらに，「缶」と「できる」の2種類のcanを区別できない．

3.3 ステップごとの分類結果の集約

各ステップの分類結果を元に，テスト・ケース全体に対するトリガーを求める．2.3節で言及した，coverage以外のごく少数のステップで分類が決まるという知見を反映して，まずcoverage以外のステップを集計する．coverageしか出現しなかったテスト・ケースはcoverageに分類する．そうでないときは，coverage以外のトリガー・タイプの出現回数に比例したスコアを付ける．例えば,10ステップからなるテスト･ケースのうち3ステップがvariation，1ステップがwork/stressに分類された場合，出力はvariationが0.75，work/stressが0.25となる．

4. 性能評価実験

自動分類手法の性能を評価するために，人手で分類したテスト・ケース・データを正解データとして正解率を求めた．全部で213のテスト・ケース，またその中に2,114のステップがあった．

テスト・ケースごとの実験結果を**表4**に，ステップごとの実験結果を**表5**にそれぞれ記した．「正解データ」は正解データ中で，「システム出力」は分類手法の出力中で，それぞ

れ各トリガー・タイプの出現数を示す．「正解数」はシステム出力と正解データが一致した個数を示す．適合率は，正解数／システム出力，再現率は，正解数／正解データで，それぞれ計算される．前者は，システム出力がどれだけ正しいかを，後者は正解をどれだけ網羅するかを示す．全データに対する適合率と再現率は一致する．なお，出現しなかったトリガー・タイプは記載していない．

テスト・ケースごとに見たとき，正解データは大部分がcoverageなので，すべての出力をcoverageとしたときの適合率77.93をベースラインとする．すると，この分類器は10ポイント以上それよりも優れた分類結果を示している．coverageの極端な偏りを考慮して，これ以外の性能も示した．この場合でも適合率，再現率の両方で70ポイント近い高い性能を示している．

一方でステップごとの性能はテスト・ケースごとの結果より低い．本手法は，正解がcoverage以外のトリガー・タイプの場合，テスト・ケースのステップ中で1つでも正しく分類できればよい．そのため，仮にステップ単位で検出漏れがあったとしてもテスト・ケースごとでは正解する可能性がある．この傾向を考慮すれば，ステップを誤ってcoverage以外に分類すると致命的なので，再現率よりも適合率が重要となるといえる．

4.1 分類を間違えたテスト・ケースの分析

間違いの例を観察すると，プロジェクト単位でテスト・ケース記述に癖があり，これが誤分類を引き起こしていることが分かった．

例えば，ある2つのプロジェクトでshould notというパターンを含むステップを分類したところ，プロジェクトAではshould notを含む22ステップすべてがvariationであった．一方，プロジェクトBでは4ステップすべてがcoverageであった．このパターンは，多くのプロジェクトではvariationに出現し，Aではその傾向通りである．例えば，… should not be available …というふうに，エラーの状態を示唆することが多い．一方Bでは逆の傾向を示している．実際の記述を観察すると，Bではcoverageのステップに，It should not popup any messageという文言が定型文として出現していた．これはプロジェクト固有の癖に影響された例である．

このように，プロジェクトごとに書き方の癖があるため，完全に汎用でシンプルなパターンを記述することは難しい．この対策としては，プロジェクトごとに使用するパターンを選別できるとよい．つまり，人手で分類を行いながらshould notというパターンがこのプロジェクトにおけるステップ分類に寄与していないことを学習できればよい．これは今後の課題である．

4.2 他言語への適用

本稿で提案した手法は単語分割と品詞推定，そして単語列へのパターンのみを利用している．そのため，パターンを言語ごとに生成する必要がある点を除けば，単語分割と品詞推定さえできれば他言語に対しても適応可能な手法である．例えば実験で利用したLanguageWareは，日本語を含む多くの主要言語に対応している．これらの言語に対しても，本実験で適用したパターンに対応する各言語用のパターンを用意することで，分類を行うことができる．

ただし，言語ごとの特性の違いがパターンに現れることは留意すべきである．例えば，否定の表現を検出したい場合，日本語では助動詞「ない」をパターンに記述すればよいが，英語の場合はnotのほかに，neverやhardlyなどの副詞で記述される場合もある．これらは手作業で言語ごとに個別に用意する必要がある．また，言語によってテスト・ケース記述の詳細度など，特性の違いがあるかもしれないので，別言語で同等の性能を達成できるかどうかは未知である．

5. 結論

効率的なテストを計画，実施するために，テスト・ケースをODCトリガー・タイプで自動分類する手法を提案し，評価実験を行った．分類は3つの処理からなる．初めにコピー･ペーストを検知して，テストと直接関係のない準備ステップを検出する．次に，各ステップの記述に対し，単語列パターンの一致を探すことでステップごとの分類を行う．最後に，分類されたステップ数に応じて，テスト・ケースの分類を決定する．この手法は，言語ごとにパターンと単語分割器が必要である点を除けば，特定の言語の特徴を利用していないため，汎用性が高い．また，パターンは簡単に記述できるのでプロジェクトに特有な特徴があれば容

表4. テスト・ケースごとの分類器の性能評価

	Coverage	Variation	Sequencing	計	Coverage以外
正解データ	166 (77.93%)	46 (21.6%)	1 (0.47%)	213	47
システム出力	167 (78.4%)	41 (19.25%)	5 (2.35%)	213	46
正解数	157	32	0	189	32
適合率	94.01	78.05	0	88.73	69.57
再現率	94.58	69.57	0	88.73	68.09

表5. ステップごとの分類器の性能評価

	Coverage	Variation	Sequencing	Work/stress	計	Coverage以外
正解データ	1,967 (93.05%)	141 (6.67%)	4 (0.19%)	2 (0.09%)	2,114	147
システム出力	1,994 (94.32%)	100 (4.73%)	19 (0.9%)	1 (0.05%)	2,114	120
正解数	1,926	72	2	1	2,001	75
適合率	96.59	72	10.53	100	94.65	62.5
再現率	97.92	51.06	50	50	94.65	51.02

易に拡張できる．実験では，人間が分類した分類結果を正解として，適合率と再現率で評価を行った．その結果，いずれも 90% 近い分類性能を示した．これはベースラインより 10 ポイント以上高い値であった．特に検出が難しいと考えられる，coverage 以外のトリガー・タイプに対しても，70% 近い分類性能を示した．

テスト作業の効率化はソフトウェア開発において，その効率化と高い品質を得るために必要である．本稿で述べた手法は効率的なテスト作業を支援するために必要な技術であり，また適用可能性の高い手法であるため，その効果は大きい．特に性能の点で高い数値を示したことで，テスト・ケース分類の自動化に基づくテスト効率化手法の適用範囲を大きく広げ，その価値を高める技術としての有用性を示すことができた．

謝辞

本研究に関して IBM Research の Howard Hess 博士から有用な助言をいただきました．ここに感謝の意を表します．

付録 A. パターン・ファイルの表現力

単語分割処理によってできた各単語は，（表層，原型，品詞）の 3 つ組みで表現される．表層は文書中に出現した文字列，原型は活用を直した文字列，品詞は単語の品詞タイプである．例えば，was という単語は，（was，be，動詞）となる．なお，LanguageWare は時制など，3 つ組み以外の情報も出力するが今回は使わない．

単語列パターンはこの 3 つ組みの列に対する正規表現として記述する．例えば，（.*，will，助詞）–（.*，not，助詞）–（.*，.*，動詞）というパターンで，「will not + 動詞」という単語列を含むかどうかの条件を表せる．

参考文献

［1］ M. Butcher, H. Munro, T. Kratschmer: “Improving software testing via ODC: Three case studies,” in IBM Systems Journal, Vol.41(1), pp.31-44,(2002).

［2］ Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray, Man-Yuen Wong: “Orthogonal Defect Classification - A Concept for In-Process Measurements,”. in IEEE Transactions on Software Engineering, Vol. 18, No. 11, (1992).

［3］ Ram Chillarege, Kothanda Ram Prasad: “Test and Development Process Retrospective – a Case Study using ODC Triggers,” in Proceedings of the International Conference on Dependable Systems and Networks, pp. 669-678, (2002).

［4］ Stephen H. Kan, 古山恒夫, 富野寿: ソフトウェア品質工学の尺度とモデル, 構造計画研究所, ISBN4-320-09743-2 (2004).

［5］ Richard M. Karp, Michael O. Rabin: "Efficient randomized pattern-matching algorithms," in IBM Journal of Research and Development, Vol. 31(2), 249-260, (1987).

［6］ LanguageWare: http://www-01.ibm.com/software/globalization/topics/languageware/index.jsp

［7］ Rational Quality Manager (RQM): http://www-06.ibm.com/software/jp/rational/products/test/rqm/

日本アイ・ビー・エム株式会社，
東京基礎研究所
副主任研究員

海野 裕也 Yuya Unno

［プロフィール］
2008 年日本 IBM 入社．東京基礎研究所において自然言語処理やテキスト・マイニングの研究に従事．言語処理学会会員．

日本アイ・ビー・エム株式会社，
東京基礎研究所
主任研究員

中村 大賀 Taiga Nakamura

［プロフィール］
1999 年日本 IBM 入社．東京基礎研究所においてサービス・ソフトウェア・エンジニアリングの研究に従事．PhD in Computer Science. IEEE，ACM，情報処理学会各会員．