文字と構造のサーチによるワンストップコード探索環境: CodeDepot

イェ　ユンウェン（(株) SRA）　中小路　久美代（(株) SRA）

概要　組織やグループで現在開発中の，あるいは既に開発したシステムやオープンソースといった大量のソースコードを，その組織の知財として活用が望まれている。そのようなソースコードに特化して検索するのが，エンタープライズ検索エンジン CodeDepot である．CodeDepot は，ソースコードを構造化された自然言語の文書とみなし，プログラムの構造情報と意味情報を同時に考慮しながら，正規化処理と重複インデキシングを行なう．これにより，ウェブインタフェースを介して数億行以上の大規模ソースコードを瞬時に検索でき，現場での実用に耐えるシステムとなっている．検索クエリ文字列のバリエーションや複雑な条件検索，さらには文字列をトークンに置換することで，コードクローンと呼ばれる構造的に類似するプログラム断片を，インタラクティブに検索できる．現時点で数十のソフトウェア開発組織で，コーディング作法の学習や保守のような開発作業の様々な局面で評価利用されている．利用した開発者からは，システムの高速性と簡便性が高く評価されている．

1. はじめに

ソフトウェア開発作業は，プログラマが自分の有する知識をプログラムのソースコードに外在化する知的な作業である[1]．作業の結果として生成されるソースコードは，知識アーティファクトであり，知識を蓄積と伝搬する媒体でもある．この点から鑑みるとソースコードは，ソフトウェア開発を営みとする組織にとって貴重な資源であり，知財である．残念なことに現状では，ソフトウェア企業が，開発中，あるいは過去に開発してきたソースコードやそのソースコードに蓄積されてきた知識を活用しているとは言い難い．

ソースコードを知財として活用するにあたっては二つの課題がある．一つ目はソースコードを蓄積すること，二つ目は蓄積したソースコードから必要とする情報を検索することである．一方でソースコードの蓄積がないと検索して利用することができない．他方で，検索が難しいと蓄積があっても活用されることがない．蓄積と検索は，その両者が同時に解決されるべき課題である[2]．

これらの課題は，ソースコードに限らず一般的な知識と情報の利用にも共通するものである．Google のようなウェブ検索エンジンが知識の蓄積と活用にもたらした影響は，大いに参考となる．従来のナレッジマネジメントのアプローチでは，蓄積の時点で知識の選別，整理と構造化といった初期投資のかかる作業が必要であった．これに対して Google といった検索エンジンは，蓄積時点での選別や整理，構造化を想定していない．これらの検索エンジンには，性能，規模，ユビキタスという三要素が関わっている．第一に，検索結果が瞬時に返されることで，情報検索という作業を，時間と労力と手間のかかる作業から単純なタスクに変える．第二に，検索対象となる情報の規模の大きさが，もともと選別と整理を経ていない情報の信頼性と有用性を保証するものとなった．第三に，検索という機能にウェブブラウザを通してアクセスできることで，情報を検索するという作業が，ユビキタスな体験として可能となった．このように検索作業が簡易化され，検索結果の有効性が増し，かつ検索行為が遍在的になることで，情報を発信する価値がネットワーク効果により増幅され，ウェブを介した情報発信を促進する．蓄積する情報と知識の量はさらに増加し，その量の増加により質も相乗的に向上する．

本論の著者ら自身の研究も含めたこれまでのソースコード検索支援に関する研究において，ソースコードを簡便に検索する機能が，ソフトウェア開発における作業の様々な側面を支援することが報告されている[3]．しかしながら，これらの研究で報告されているツールは研究プロトタイプであり，上述した性能・規模・ユビキタスという三つの要素を満たすようなソースコードを検索する実用的なシステムは未だ存在していない．

本論で解説する CodeDepot システムは，企業で開発中のプログラムをはじめ，過去に開発したシステムやオープンソースといった大量のソースコードを，ウェブインタフェースを介して統一的に，高速に検索するものである．CodeDepot は，ソースコード検索技術がソフトウェア開発に与える影響や効果を探究したり評価したりするために開発したものではない．一つの企業が生産する

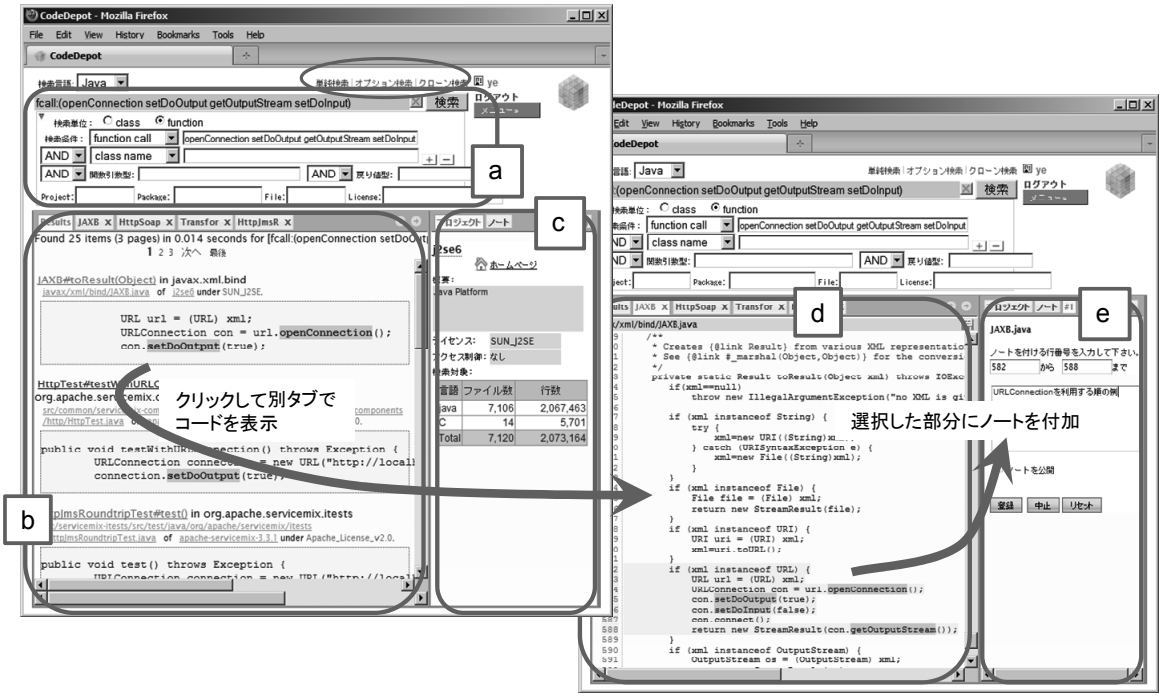

図 1. CodeDepot のユーザインタフェース

大量のソースコードを資産として高速に検索するための，実用的な製品として開発したものである．具体的には，

(1) 大規模（数億行以上）のソースコードを瞬時に検索できること（高性能による検索作業の簡便化）

(2) 蓄積にかかる初期投資を最小限に抑えられる，すなわち蓄積時に選別や整理を必要とせず，検索対象の規模の大きさで信頼性と有用性を保証できること（規模の大きさによる検索結果の有効性の増加）

(3) コード再利用に限らず，プログラミングから保守や品質管理に至るまでの，ソフトウェア開発と管理における様々な検索タスクを統一したウェブインタフェースで提供すること（検索行為の遍在化）

(4) 検索結果にメモを付加することにより，検索結果のソースコードに関する知識を漸次的に蓄積していけること（検索とノウハウ蓄積の一体化）

を目指して，ソースコードに特化した大規模な高速の検索技術の開発を目指した．

これらの目標を実現するために，我々はソースコードのリッチな構造情報と全文検索の機構を統合するシステムを構築した．ソースコードを構造化された自然言語の文書とみなし，プログラムの構造情報と意味情報を同時に利用する．grep のような手軽さを保ちつつ，正規化処理によって，検索クエリとなっている文字列のバリエーション（たとえば空白やコメント文やキャメルケースで書かれた関数名）の検索結果を同時に表示できたり，ソースコードの構造情報を利用して，ある文字列を関数名に含み特定の入力タイプや出力タイプを持つ関数を検索できたりする．さらに，文字列をトークンに置換することで，類似構造を持つコード断片，いわゆるコードクローンを，インタラクティブに検索することができる．

CodeDepot はウェブブラウザによるユーザインタフェースを備えており，ソースコードに関する様々な検索をワンストップで提供する（図 1）．単純な文字列検索から高度な検索条件式の記述まで対応するシンプルな入力インタフェースと，個々のフィールドを指定しながら検索条件を構造的に入力していけるようなオプション検索インタフェース，および，類似構造を持つプログラムを検索するためのクローン検索インタフェースを備えている（図 1a）．検索結果をインタラクティブに絞り込んでいけるように，検索結果リストから，それぞれのソースコードをタブ形式で複数個表示していくことができる（図 1b,d）．また，表示されたソースコードの一部に「ノート」と呼ぶメモを付加し蓄積することができ，自分自身のため，あるいは他の開発者のために，ソースコードを改変することなく，ソースコード内の指定した箇所に対してメモを貯め共有していくことができる（図 1e）．

以下に，2 章ではコード検索支援技術の現状を述べる．3 章では CodeDepot の概要とアーキテクチャを説明し，高速検索を実現するための実装方式について解説する．4 章では CodeDepot の定量的な性能評価と利用実例を通しての定性的な性能評価を説明し，5 章で本論を結ぶ．

2.開発におけるコード検索の支援

プログラマは，修正する箇所やその波及効果を調べる

とき，バグを引き起こす箇所を探るとき，あるいは開発済みまたはオープンソースから既存機能を再利用するとき，といった多様な局面で，一つのシステム内や，複数のシステム間で，ソースコードを検索する [4]．開発者が行っている作業のうちの約 30%の時間が，ソースコードの検索と調査に費やされている [5].

これらの検索活動で使われているツールの代表は，grep であろう．しかし grep は，そもそもソースコードの構造を考慮したり，大規模な複数システムの横断的な検索に適したものではない．grep 以外にも，保守支援ツールや Eclipse といった開発環境には，ソースコードの構造を利用した検索ツールが装備されていることが多い．しかしながら，これらのツールの多くはシステムの完全解析を前提としておりソースコードに修正が加えられる度に再解析の必要があったり，単一のシステムのみを対象としておりシステムや言語をまたがった検索が容易に行えなかったりする．複数のシステムのコードを横断的に検索する研究ツールとしては，プログラムのフィンガープリントによる検索を行える Sourcerer[6]や，クラスのインタフェースによる検索を試みる Merobase[7]，あるいはコンポネントの呼び出し関係を利用して検索結果をランキングする Spars-J[8]などが挙げられる．これらは，プログラムの構造情報の一部を利用した検索方法の有効性を検証するために開発された研究プロトタイプである．

gonzui（http://gonzui.sourceforge.net/）は，複数システムのコードを横断的に検索する先駆的な検索エンジンであり，関数名やコメントに限定した検索を行える．Google の codesearch（http://google.com/codesearch）は，強力な正規表現による検索機能が主な特徴であり，クラス，関数，ファイル名などに分けて検索することができる．Koders（http://www.koders.com/）では，クラス名，関数名，関数呼び出しに絞って検索することができ，Krugle（http://www.krugle.org/）は，コメント，コード，クラス定義，関数定義，関数呼び出し毎の検索や，厳密な類似コードの検索など，充実した検索機能を提供している．これらのシステムは，ユーザが検索要求として記述した条件を満たす情報でさえあれば，どのソースの情報であるかを問わず検索結果として返す，という，一般的検索エンジン（general search engines）のデザイン原則を踏襲したものである．

我々は，ソフトウェア開発におけるコード検索ニーズの多くは，検索対象に関しては記憶が曖昧な部分（例えば正確なファイル名を覚えていない，など）が多いものの，検索結果として求める情報は明確である（列挙してもらえばこの中からどれが探していたものかがわかる），といった状況で生じる場合が多いと考えている [3]．大量のソースコードを組織の知財として活用するための，ソースコードのためのエンタープライズ検索エンジンとしては，開発者の記憶に残っている部分的な情報を利用した複数の検索条件の組み合わせを柔軟に行えることに加えて，複数プロジェクトや複数パッケージに限定した検索や，プログラムのシグネチャ情報を利用した検索，あるいはファイル単位ではなく関数単位の検索，さらにはユーザのアクセス権限の管理などといった機能が望まれる．

3. CodeDepot システム

CodeDepot は，大量なソースコードを，単一のインタフェースで高速に検索できる検索エンジンと，開発者の状況とニーズに合わせて小気味よく動く，シンプルなインタフェースを実現したものである．

3.1 CodeDepot 概要

CodeDepot は，コードリポジトリに登録した，システムやプロジェクトの全ソースコードを対象として検索するウェブアプリケーションである．Java で実装したウェブサーバ側で，リポジトリの管理および検索のためのインデックス作成と検索制御を行う．ソースコードは，検索結果をハイライト表示するために HTML 形式に変換され保存されている．ユーザとしての開発者が操作する検索インタフェースはウェブブラウザであり，Javascript で実装されている．現在検索できる言語は，Java および C/C++である．

サーバに登録するソースコードとして想定しているのは，(1)開発中のプロジェクトのソースコード，(2)オープンソースで公開されているプロジェクトのソースコード，(3)組織内でこれまでに開発してきたソースコードと保守中のソースコード，などである．これらを一括して登録しておくと，これらの全ソースコードを横断的に検索することができる．

3.2 CodeDepot の機能

CodeDepot は，既存のソースコード検索システムが実装している下記の検索機能を備えている．

(1) 自然言語による意味検索．例えば，クレジットカード番号の正しさをチェックするプログラムを検索するために“credit card valid”で検索すると，
valid_card, creditCardValidator, isValidCreditCard, validateCreditCard, validCreditCard,

creditCardValidation

といった名前の関数が返される．正確な名前を知らなくても検索ができる．

(2) プログラムの構造情報を利用した検索．パッケージ名，ファイル名，クラス名，関数名，関数呼び出し，コード，コメントといったプログラムの構造エレメントに限定したり，あるいは除外したりして検索を行う．

さらに CodeDepot では，下記の機能を実装している．

(1) ファイル単位だけではなく，クラス単位と関数単位での検索．

(2) 複数のプログラム構造情報の組み合わせ検索と複数のプロジェクト，パッケージ，ファイル，ライセンスの組み合わせ条件による検索結果のフィルタリング．

(3) コード断片と，三つの厳密度レベルで類似する構造をもつプログラム（コードクローン）の検索．

(4) 版管理システム（CVS あるいは Subversion）との連携による検索対象の自動更新．日々更新される開発中のプロジェクトのソースコードと，バッチ処理されている既存システムのソースコード（たとえば開発済みのプロジェクトや，利用しているオープンソースのソースコード)が，区別なく同一のインタフェースで検索できる．

(5) 登録されたシステムあるいはプロジェクト毎に，許可されたユーザだけが検索できるようなアクセス権の設定．インデックスされたコードは，パブリックなものとアクセス制限があるものとに分けられ，前者はすべてのユーザの検索対象となるが，後者はそのアクセス権を有するユーザにのみ検索対象となる．

(6) 検索結果として検索されてきたソースコードを見ながらわかったこと，気づいたことなどを，「ノート」としてソースコードの選択した箇所に付加．プログラムに直接挿入されるコメント文とは異なり，元のソースコードを改変しないで，プログラマが任意に選択したコード箇所にノートを付加し保存，共有することができる．コードを表示する際には，付加されたノートも同時に表示される．

3.3 CodeDepot のシステム構成

CodeDepot は以下の部分から構成されている(図 2).

(1) コードリポジトリ：HTML 化したソースコードを格納する．ユーザの追加したノートも，ソースコードとリンクした形式で格納されている．

(2) バッチ処理インデックス作成部：コードの更新がないソースコードをインデックスする．

(3) 自動更新インデックス作成部：開発中のシステムを版管理システムから自動的にチェックアウトし，変更した部分のみについて，インクリメンタルにインデックスを更新する．

(4) 語検索インデックス DB:語検索のためのインデックスを格納する．

(5) クローン検索インデックス DB:クローン検索のためのインデックスを格納する．

(6) 検索制御部：ユーザ毎にアクセス権の有無を確認し，パブリックなものとそのユーザがアクセス権をもつコードのみを検索結果として表示する．

(7) 検索インタフェース：ウェブブラウザによる Ajax 的な検索インタフェースを提供する．

(8) 管理登録インタフェース：ウェブブラウザ上で，システム，プロジェクトやファイルの登録と管理，ユーザ管理，およびアクセス権管理を行う．

3.4 CodeDepot のユーザインタフェース

3.4.1 ユーザ操作の流れ

CodeDepot のユーザは，ウェブブラウザを利用して検索を行う．検索インタフェースは，検索条件を入力する領域（図 1a），検索結果と各コードを表示するタブベースの領域（図 1b,d），プロジェクト情報とノートを表示するタブベースの領域（図 1c,e）から構成される．

検索条件を入力し，検索ボタンをクリックすると，検索結果がリストとして表示される（図 1b）．リストの各アイテムは，クラス名または関数名と，そのパッケージ名，プロジェクト名，プロジェクト内のファイルパス，ライセンス，および検索条件にマッチしている箇所の数行の抜き書きから成る．リスト内のアイテムは，検索キーの重要性の違いなどを考慮して計算した適合度のスコアによりランキングされている．アイテムは 10 個ずつ別ページに表示され，検索結果をページごとにブラウズしていくことができる．

検索結果リストの中からクラス名または関数名をクリックすると，そのソースコードのファイルが，新たなタブのページとして表示される（図 1d）．検索要求に合致

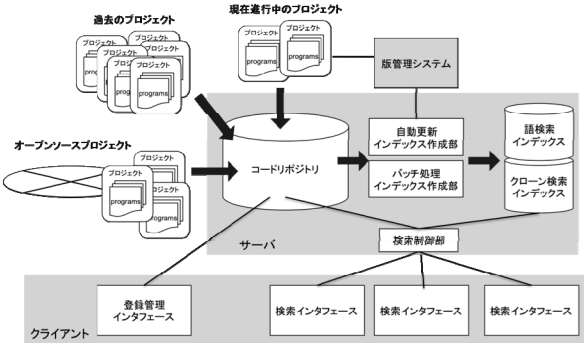

図 2 CodeDepot のシステム構成

している箇所がハイライトされ，その該当箇所までスクロールした状態で表示される．また，検索結果リストの中の各アイテムのパッケージ名をクリックすると，このパッケージ名を検索条件として含むか／除外するか，を明示的に選択できるメニューが表示される．これによって，検索結果をインタラクティブにフィルタリングしながら絞り込むことができる．

右側の領域のプロジェクトタブには，選択した検索結果のプログラムが属するプロジェクトの名称やURL, 概要，ライセンスやアクセス権，言語，ファイル数や行数といった情報が表示される（図 1c）．また，ノートタブを利用して，コード表示タブに表示されているソースコードをマウス選択した部分に，ノートを付加することができる（図 1e）．ノートが付加されると，コードを表示した際，左端の行番号にノートアイコンが表示されるようになる．以後，そのコードを表示するたびに，付加されたノートがノートタブに表示される．

3.4.2 検索条件入力部

CodeDeopt は, 検索条件を入力するためのインタフェースとして，「オプション検索」と「単純検索」という，入力した検索語をサーチする二種類のインタフェースと，「クローン検索」という類似したプログラム構造を検索するためのインタフェースとを提供している．これら計三種類のインタフェースは，CodeDepot ウィンドウ上部のボタンで切り替えることができる（図 1a の上部）．

オプション検索インタフェース（図 1a）は，検索語を入力するフィールドと, プロジェクト名, パッケージ名，ファイル名およびライセンスを指定するフィールドとから成る．検索語を入力するフィールド間には，AND または OR の選択があり，検索条件を自由に組み合わせることができる．フィールドは随時追加していくことができる．これらの組み合わせにより，「名前が random string を含み，引数として 2 個の int をとり，戻り値として string を返すような, Apache License v2.0 の関数」といった検索を行うことができる．

単純検索インタフェースは，単一のテキストウィンドウにテキストで検索条件を記述するようなインタフェースである．初心者のユーザ向けに，簡単な検索語，たとえば，“random string”を入力すると，ランダムなストリングを生成する機能を持つクラスを見つけることができる．同時に，システムの機能とフィールドの名を熟知しているようなユーザ向けに，複雑なロジック式的な検索式を入力して検索することができる．たとえば，図 1a の検索は，下記のクエリを単純検索インタフェースに入力するのと同等である．

```
fcall:(openConnection setDoOutput getOutputStream
setDoInput) AND unit:method
```

CodeDepot を利用していくうちにオプション検索を利用するユーザが徐々にこのような検索式の書き方を学習していけるように，オプション検索のインタフェースの最上部には，ユーザの入力した検索要求をロジック式として展開し表示している（図 1a 上部のグレーの背景がついた部分）．

クローン検索インタフェースでは，入力ウィンドウにプログラム断片を記述する．図 3 は，freebsd-7.2 の usr.bin/edquota/edquota.c というファイルにある hasquota という関数の 891～897 行目を入力条件として記入し, クローン検索を行った例である. CodeDepot は，ここに記入されたコードと類似する構造を有する，C/C++，あるいは Java のプログラムを検索し，検索結果をリストする．類似度は，低レベルから高レベルまで三段階の厳密度で判定する．高レベルでは，コメントや識別子の差異などを除いて完全に一致するコードを, 中，低レベルの厳密度では，それぞれ差分が一行，もしくは二行以内でその他は構造が完全に一致するコードを，検索する．

3.5 CodeDepot の実装

検索エンジンを実装する技術の中核はインデキシング機構である．ここでは CodeDepot のインデキシングのメカニズムを説明する．CodeDepot では，検索語をキーとしてサーチする検索と，類似する構造のコードをサーチするクローン検索のために，それぞれ異なるインデキシングを行っている．

3.5.1 語検索インデキシング

CodeDepot では，高速な検索を実現するために，様々な検索要求と方法に対応したインデキシングを行っている．インデキシングに先立って，ソースファイルを解析し，複数のインデックスユニットを生成する．さらに，各ユニットをプログラムの構造に沿って解析し, コード，コメント，プロジェクト名，パッケージ名，関数名，といったフィールド属性を同定する．

コードに限定した検索，コメントに限定した検索，コードとコメントを同時に検索といったあらゆる場面で高速に検索するために, CodeDepot では重複インデックスファイルを作成する．また，クラス単位と関数（メソッド）単位の双方で検索を行えるように，クラス単位と関数単位の両方を含むインデックスを構築する．さらに，

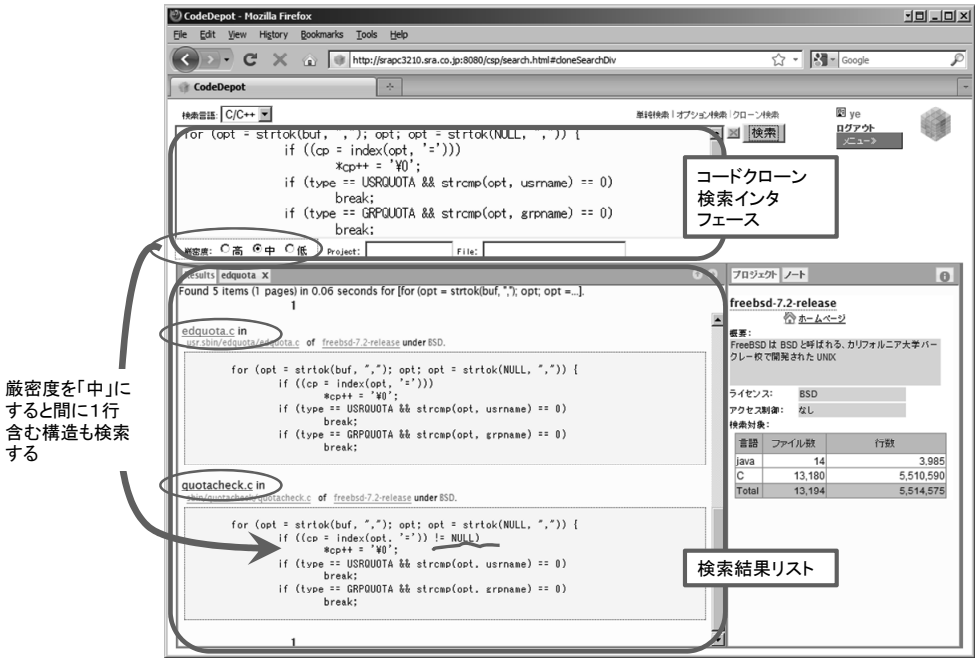

図 3. クローン検索インタフェースと検索結果

同じフィールドに対して様々な形式で検索要求が生じることを想定し，一つのフィールドに複数のインデックスを作ることにより検索の高速化を図っている．入力タイプと出力タイプにもこれに特化した正規化処理を行い，Java のクラスを完全修飾クラス名（FQCN: Fully Qualified Class Name）でも，単純名（simple name）でも検索できるようにインデキシングしている．

インデックスのフォーマットは全文検索の倒置インデックスを利用している．上記の前処理した結果を，通常の自然言語文書と同様に，一般的な検索エンジンの全文検索技術を利用してインデックスファイルを構築する．CodeDepot は，インデックス作成と検索のコア機能を提供する Lucene [9]と，検索のウェブインタフェースと検索を高速化するためのキャッシュ機構を提供する Solr [10]とを利用して実装されている．

3.5.2 クローン検索インデキシング

類似した構造のプログラム断片（コードクローン）を同定する研究ツールの多くは，ソースコードを文字列（厳密にいうとトークンシーケンス）とみなして，文字列の最長共通部分文字列を求めるアルゴリズムやデータ構造に基づき，バッチ処理的にクローンを検出する[11]．これに対して CodeDepot は，ソースコードをワードから構成するテキストとして処理することによって，大規模なソースコードリポジトリからインタラクティブにコードクローンを検索することを可能にしている．

たとえば，“if (a>b) m=a; else m=b;” というコード断片があったとする．このコード断片は，トークンシーケンスでは次のように変換される．

if	(	$	>	$	)	$	=	$	;	else	$	=	$	;

CodeDepot では，同じコードを下記のようなワードシーケンスとして変換する．

if($>$)	$=$;	else	$=$;

ここでいうワードは，プログラムの意味を表現するものではなく，プログラムの様々な構造を抽象した基本要素となる．たとえば“if($>$)”は，二つの変数の greater-than の比較による if 文を抽象した文法構造である．

自然言語と対比すると，トークンは文字で，ワードは文字からなる語彙である．語彙列の長さは文字列の長さより短くなるため，クローン処理の効率が上がりスケーラビィリティも向上する．CodeDepot では，プログラムをワードテキストに変換し，変換されたワードテキストをキーとする倒置インデックスファイルを作成する．このインデキシング処理は，前節で説明した語検索インデキシング処理と並行して行う．

クローン検索を行う際には，検索要求となるコード断片をワードテキストに変換し，作成したクローン検索インデックスファイルを利用しながら，指定した厳密度に応じて類似構造をもつプログラムを同定する．

4. CodeDepot の性能と機能評価

二種の実験により CodeDepot の技術目標に関して性能評価を実施した．

性能評価には，Dual-Core AMD Opteron 1GHz の CPU と 6GB のメモリを擁する CentOS を用いた．まず，約 25 万ファイル，計 6800 万行あまりのソースコードに

```
URL url = new URL("http://10.0.1.2/search");
URLConnection conn = url.openConnection();
conn.setDoOutput(true);
OutputStreamWriter wr = new
   OutputStreamWriter(conn.getOutputStream());
wr.write("q=string"); wr.flush(); wr.close();
conn.setDoInput(true);
BufferedReader br = new BufferedReader(new
    InputStreamReader(conn.getInputStream()));
String line;
while ((line=br.readLine()) != null){
    System.out.println(line);  }
```

図 4：自分で書いたプログラム

```
Exception in thread "main"
java.lang.IllegalStateException: Already
connected
 at java.net.URLConnection.setDoInput(Unknown
Source)
 at example.URLEx.main(URLEx.java:18)
```

図 5：実行時のエラーメッセージ

対して図 1a に示す条件を指定して検索した結果，検索に費やした時間は，14 ミリ秒であった．次に，約 5 億行の Java コードをインデキシングしたリポジトリに対して，25 個の検索クエリを実行した結果，単語単位の検索に要した時間は平均 50 ミリ秒，複数の検索要求を組み合わせた場合は平均 900 ミリ秒であった．

一般に，システムのレスポンスタイムが 100 ミリ秒以下であればシステムが即時に反応しているとみなされ，1 秒以内であればシステムのレスポンスがユーザの思考のフローを中断することはないとされている[12]．この標準と照合すると，実験により計測した数値は，目標としていた技術目標を達成しており現場での実用に十分に耐えるシステムとなっていると考えられる．

語検索に比べるとクローン検索は若干検索にかかる時間が増加する．これまで報告されている既存のクローン検出システムでは，百万行のコードからクローンを検出するために 30 分以上の時間がかかるとされている[11]．これに対して CodeDepot では，6,800 万行のコードに対して，厳密度を高に指定した類似構造の検索時間は平均 1.5 秒であった．既存ツールと比べると十分に高速ではあるが， 1 秒未満となるよう改善すべきと考えている．

CodeDepot のもう一つの目標は，統一したインタフェースでプログラマの抱えている様々な検索タスクを遍在的に支援することである．CodeDepot は，現時点で数十のソフトウェア開発組織により利用評価を頂いているところである．知財としてのソースコードが検索対象となっており，各社の開発現場での利用状況を把握して定量的に分析することはできないが，回収したアンケートでは，CodeDepot の高速な検索性能と簡易なプロジェクト登録機能が高く評価されていた．また今後要望する機能としては，C/C++言語の Makefile の情報を利用したうえでコードをインデックスするものが最も多くあった．下記に，実際の開発現場のどのような場面で CodeDepot を利用したかの実例を紹介することで，大規模なソースコードの高速な検索機能がもたらす開発支援の様々な可能性を定性的に説明する．

(1) 類似するプログラムを見ながらコードのデバッグをする． 検索サイトにキーワードを送り検索結果を返すような Java プログラムを URLConnection クラスを使って図 4 のようなコードを書いていた．実行すると図 5 のようなエラーが出る．エラーメッセージを見てもどこに問題があるのかが分からない．CodeDepot を用いて，自分のコードで使っている四つのメソッドを検索条件としてサーチしてみたところ（図 1a 参照），25 件の検索結果が返ってきた（図 1b）．検索結果の 1 個目と 2 個目をタブで開いて自分のコードと比べてみると（図 1d），双方のプログラムにおいて，どうもこれら四つのメソッドの順序が自分のコードと違うことに気づいた．自分のコードでは，openConnection→setDoOutput→getOutputStream→setDoInput となっているが，検索した二つのプログラムではいずれも，openConnection→setDoOutput→setDoInput→getOutputStream という順序となっている．そこで，自分のコードでも同じように getOutputStream と setDoInput の順序を入れ替えてみたところ，エラーメッセージが出なくなった．

実際のところ URLConnection を使用する際には，データを送受信する設定の setDoOutput と setDoInput は，getOutputStream で接続を行うよりも先に呼び出される必要がある．ところが，これに関する説明は API ドキュメントには明示的には示されていない．CodeDepot によって，同じ関数を呼んでいるプログラムの例をみることでエラーが解決された事例であり，API のドキュメントの不在や不備といった課題を解消する例でもある．

(2) ありがちなミスを見つける． C 言語では，代入文を判定文とする構文が許されていることもあり，これが典型的なタイプミスとなって正しく動作しないことがある．たとえばしばしば犯すミスとして，

if (a=b) --- (1)

がある．意図しているのは，

if (a==b) --- (2)

であるにも関わらず，(1)を書いてしまう場合はよくある．

CodeDepot のクローン検索で，検索対象を C 言語を指定して，openafs-1.4.11 のプロジェクトに対して

“if (a=b) {”

で検索してみたところ，図 6 のような結果が帰ってきた．検索結果の上から三つめの sascnvldb.c のコードは，判

定式の中で定数を代入しており，ソースコードを見てみると，これは明らかにバグであることがわかった．

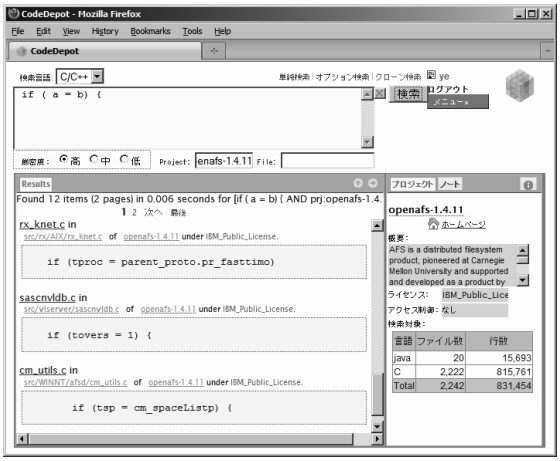

図 6：“if (a=b){“をクローン検索した例

(3) ファイルポインタ不足を引き起こす可能性のあるソースコードを一気に特定する．多くの OS では同時にオープンできるファイルの数に制限がある．プログラム中でファイルを open して read や write をする際，プログラム実行中に不要となったファイルは通常は OS が自動的に close してくれるため，プログラム中で明示的に close する必要はない．しかしシステムの規模が大きくなると，ファイルポインタが不足しエラーが生じるため，ファイルを open したまま close していない関数に close 文を追加する必要がある．CodeDepot で，検索範囲をプロジェクトとし，検索対象を関数に指定して，“fcall:(open (read OR write) -close) prj:prj1 unit:mfdef” で検索を行った結果, close を挿入すべき箇所を容易に同定することができた．

5. 終わりに

プログラムは，計算機に実行させるための構造を有すると同時に，開発者に読んでもらうための意味を持つ文書でもある．CodeDepot は，(1)プログラムを構造と意味情報を持ち合わせた構造化した自然言語の文書とみなして既存の汎用的な全文検索エンジンを利用すること，(2)ユーザである開発者がどんな検索要求を欲するか，どんな状況で検索するかといったことを分析し，リッチなインデックスを作成すること，という二点により，数億行といった大量なコードを瞬時に検索することを可能とした．CodeDepot によって，知識の宝庫であるソースコードを，組織がより効果的に活用するきっかけとなればと考えている．

参考文献

1) 大平雅雄，イェ ユンウェン，中小路久美代，山本恭裕:ソフトウェア開発における知識コラボレーション，人工知能学会誌, Vol.26, No.1, pp.66-78 (2010).
2) Ye, Y. and Fischer, G: Reuse-Conducive Development Environments, Journal of Automated Software Engineering, Vol. 12, No. 2, pp. 199-235 (2005).
3) Ye, Y. and Fischer, G: Information Delivery in Support of Learning Reusable Software Components on Demand, Proc. of International Conference on Intelligent User Interface, pp.159-166 (2002).
4) Ko, A.J. and B. Myers: An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks, IEEE Transactions on Software Engineering, Vol.32, No.12, pp.971-987 (2006).
5) Singer, J., et al.: An Examination of Software Engineering Work Practics, Proc. of 1997 Conference of the Centre for Advanced Studies on Collaborative Research, 21pages (1997).
6) Bajracharya, S., et al.: Sourcerer: A Search Engine for Open Source Code Supporting Structure-Based Search, Proc. of 2006 International Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 25-26 (2006).
7) Hummel, O., W. Janjic, and C. Atkinson: Code Conjurer: Pulling Resuable Software out of Thin Air, IEEE Software, Vol.25, No.5, pp.45-52 (2008).
8) Inoue, K., et al.: Ranking Siginificance of Software Component Based on Use Relations, IEEE Transactions on Software Engineering, Vol.31, No.3, pp.213-225 (2005).
9) Lucene: http://lucene.apache.org/
10) Solr: http://lucene.apache.org/solr/
11) Kamiya, T., S. Kusumoto, and K. Inoue: CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code, IEEE Transactions on Software Engineering, Vol.28, No.7, pp.654-670 (2002).
12) Nilesen, J.: Usability Engineering, Morgan Kaufmann, 1993

イェ ユンウェン（非会員）
E-mail: ye@sra.co.jp
1987 年中国復旦大学コンピュータサイエンス学部卒業．1990 年同大学院修士課程修了後，助手．1993 年(株)SRA に入社．2001 年米国コロラド大学より Ph.D. 2001～08 年コロラド大学客員研究員．現在 SRA 先端技術研究所チーフリサーチャ．専門は，ソフトウェア開発支援，検索エンジン，知識共有．

中小路 久美代（正会員）
E-mail: kumiyo@sra.co.jp
1986 年大阪大学基礎工学部情報工学科卒業後，株式会社 SRA 入社．現在同先端技術研究所リサーチディレクタ．1993 年米国コロラド大学より Ph.D. 1994 年米国コロラド大学認知科学研究所客員助教授，1995 年奈良先端科学技術大学院大学客員助教授，2002 年東京大学先端科学技術研究センター特任教授．専門は，ナレッジインタラクションテサイン，知的創造活動支援，ソフトウェア開発支援．

投稿受付：2010 年 9 月 3 日
採録決定：2011 年 2 月 16 日
編集担当：小島啓二（日立製作所）