頻出ソースコードパターン探索のための一手法

東京工芸大学工学部コンピュータ応用学科
宇田川　佳久

目　次

1. 研究の背景と概要

- ソフトウェア開発において、ソースコードのコピー＆ペーストが一般的に行われている.
  理由：開発コストを低くできる.
- 実務で使われているソースコードの 7%～ 23% は重複したコードである [1][11]
- 重複したコードは、ソフトウェアの保守に有害である[10][12][13]
  - 例：特定のコードシーケンスでバグが発見されたとき、バグ修正のために、類似するコードシーケンスをすべてチェックする必要がある
    ⇒ 類似するコードシーケンスを検索する機能が必要

1. 研究の背景と概要

実務面での応用

①バグあるいは脆弱なコードを探す
（品質の高いシステムを開発する）

②プログラム理解を支援する
（第3者が書いたコードを保守する）

③特定のメソッドを使ったコードを探す
（テクニックの向上）

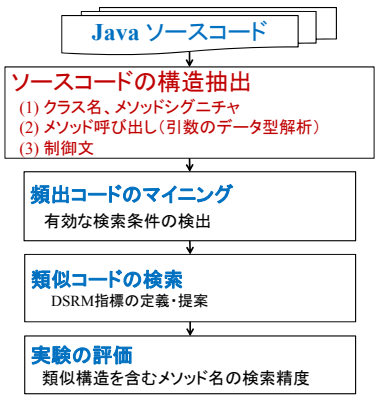
1. 研究の背景と概要

ソースコードの類似検索手法

① テキストに基づく検索
　–変数名の違いに影響を受ける
② トークンに基づく検索
　–変数名の違いは除去できるが、局所的な検索に留まる
③ メトリクス基づく検索
　–メトリクスを使った間接的な検索に留まる
④ 構造に基づく検索
　–文法に沿った検索ができるが、処理が複雑

この分野のサーベイについては、[13] をはじめとして多数ある

1. 研究の背景と概要



ソースコードの構造抽出 (1/2)

1. クラス名、メソッドシグニチャ
   <Calss Name>::<Method Signature>
   <Calss Name>:<Inner Calss Name >:<Method Signature>
   <Calss Name>:<Anonymous Calss Name >:<Method Signature>
2. メソッド呼び出し（引数のデータ型解析）
   <Variable>.< Method Name> は、データ型の解析を行い <Type>.< Method Name> に変換

これにより、変数名の影響を排除している

ソースコードの構造抽出 (2/2)

3. 制御文
   1. if（else, else if）
   2. switch
   3. While
   4. do while
   5. For
   6. break
   7. continue
   8. return
   9. throw
   10. synchronized
   11. try（catch, finally）

実験対象としたソースコード

Apache-Tomcat 7.0.42

Number of Java Files	----	1,100
Number of Classes	----	1,681
Number of Methods	----	10,640
Number of Code Lines	----	177,724
Number of Total Lines	----	334,457

Apache-Tomcat 7 は、実務レベルでも大規模ソースコード

抽出例 (1)

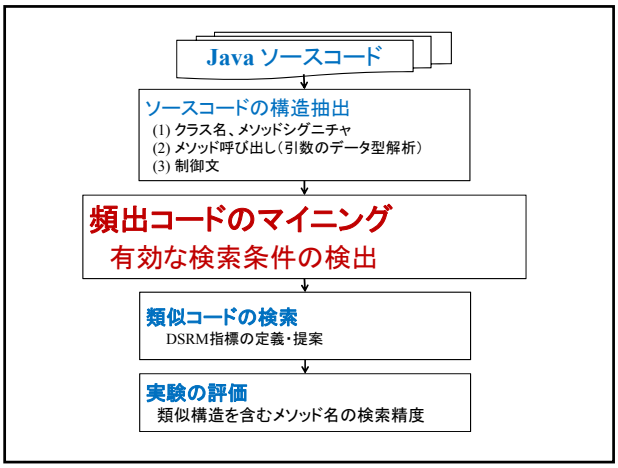
```
SSIServletExternalResolver::getAbsolutePath()
{
        SSIServletRequestUtil.getRelativePath
        getPathWithoutFileName
        if{
        IOException
        }
        RequestUtil.normalize
        if{
        IOException
        }
return
}
```

抽出例 (2)

```
InputBuffer::skip(long n)
{
    if{
    IOException
    }
    if{
    IllegalArgumentException
    }
    while{
        if{
        CharChunk.setOffset
        }
        else{
        CharChunk.getLength
        CharChunk.setOffset
            if{
            CharChunk.getChars
            }
            else{
            }
        realReadChars
            if{
            break
            }
        }
    }
return
}
```

抽出例 (3)

```
SecureNioChannel::read(ByteBuffer dst)
{
    if{
    IllegalArgumentException
    }
    if{
    return
    }
    if{
    IllegalStateException
    }
    sc.read
    if{
    return
    }
    do_while{
    ByteBuffer.flip
    SSLEngine.unwrap
    ByteBuffer.compact
        if{
        SSLEngineResult.bytesProduced
            if{
            tasks
            }
            if{
            break
            }
        }
        else_if{
        break
        }
        else{
        IOException
        }
    }
return
}
```



頻出コード v.s. 頻出集合

本研究で開発したアルゴリズム

```
/* Mining Frequent Sequences Algorithm */
LS= Φ;
k=1;
T_k = { i | i ∈{Control Statements} };
repeat
    k = k+1;
    C_k = Retrieve_Cand( MS, T_k-1 );
    T_k = Φ;
    for ( each c ∈ C_k ) {
        if ( at least one element of c is not
             a control statement
             && |c| ≧ N × minSup) {
            LS.add(c);
            T_k.add(c) };
        }
    }
until T_k = Φ;
Result = LS;
```

Apriori アルゴリズム

```
/*  APRIORI Algorithm  */
LS= Φ;
k= 1
F_k = {i | i ∈ {frequent items of length 1} };
repeat
    k = k+1;
    C_k = apriori_gen(F_k-1 );
    T_k = Φ;
    for ( each c ∈ C_k ) {
        if (  c ∈ T

            && |t| ≧ N × minSup ) {
            LS.add(c);
            F_k.add(c) };
        }
    }
until F_k = Φ;
Result = LS;
```

頻出コード抽出アルゴリズムの性能

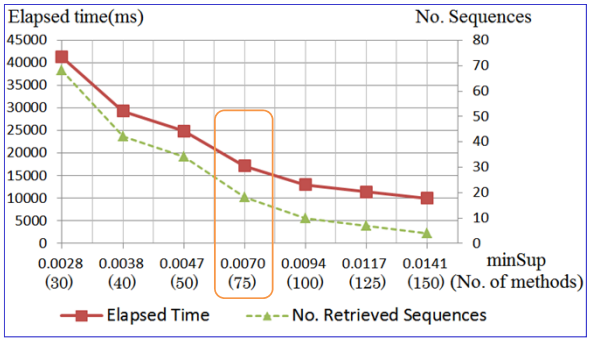
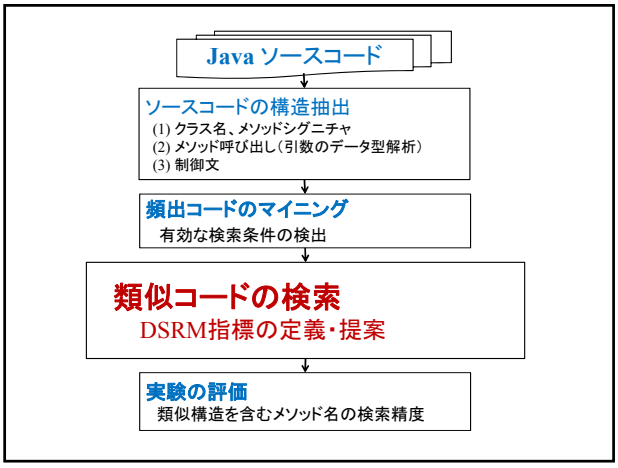


Fig.2 Number of retrieved sequences and elapsed time for each minSup

実験結果：頻出シーケンスの一覧

minSup= 0.0070

No	Sequences	Number of occurences
1	if{ → Log.debug	267
2	if{ → Log.debug → }	259
3	if{ → StringBuilder.append	218
4	catch{ → Log.error	150
5	catch{ → ExceptionUtils.handleThrowable	130
6	if{ → IllegalArgumentException	127
7	if{ → IllegalArgumentException → }	127
8	catch{ → Log.error → }	112
9	if{ → org.apache.juli.logging.Log.debug	101
10	if{ → StringBuilder.append → }	100
11	if{ → org.apache.juli.logging.Log.debug → }	99
12	if{ → StringBuilder.append → StringBuilder.append	97
13	if{ → IOException	95
14	if{ → IOException → }	95
15	catch{ → MBeanException	94
16	catch{ → MBeanException → }	94
17	if{ → StringBuilder.append → StringBuilder.append → }	92
18	catch{ → Log.error → } → }	75



類似コードの検索

Sørensen-Dice 指標

$S_{Sorensen\text{-}Dice} = 2a / (2a + b + c)$

a = 集合 A と B に共通する要素数
b = 集合 A だけに属する要素数
c = 集合 B だけに属する要素数

$$S_{Sorensen-Dice}(\mathbf{A},\mathbf{B}) = \frac{2|\mathbf{A} \cap \mathbf{B}|}{2|\mathbf{A} \cap \mathbf{B}| + |\mathbf{A} \cap \neg\mathbf{B}| + |\neg\mathbf{A} \cap \mathbf{B}|}$$

3個の集合への拡張

$$S_{Sorensen-Dice}(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{3|\mathbf{A} \cap \mathbf{B} \cap \mathbf{C}|}{3|\mathbf{A} \cap \mathbf{B} \cap \mathbf{C}| + 2|\mathbf{A} \cap \mathbf{B} \cap \neg\mathbf{C}| + 2|\mathbf{A} \cap \neg\mathbf{B} \cap \mathbf{C}| + 2|\neg\mathbf{A} \cap \mathbf{B} \cap \mathbf{C}| + |\mathbf{A} \cap \neg\mathbf{B} \cap \neg\mathbf{C}| + |\neg\mathbf{A} \cap \mathbf{B} \cap \neg\mathbf{C}| + |\neg\mathbf{A} \cap \neg\mathbf{B} \cap \mathbf{C}|}$$

N個の集合への拡張

$$S_{\text{Sorensen-Dice}}(X_1, X_2, \ldots, X_n) = \frac{n|X_1 \cap X_2 \cap \cdots \cap X_n|}{\sum_{r=0}^{r=n-1} |(n-r)\ SetComb(X_1 \cap X_2 \cap \cdots \cap X_n,\ r)|}$$

SetComb generates a combination of r-sets with the negation symbol from n-sets.

N個の要素から成るシーケンスの類似度（DSRM）

Where m ≥ n

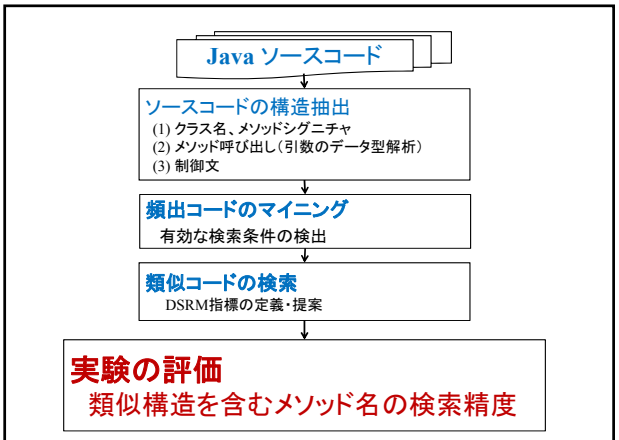
$$Sim_{DSRM}([S_1 \to S_2 \to \ldots \to S_m], [T_1 \to T_2 \to \ldots \to T_n]) = \frac{n \mid [S_1 \to S_2 \to \cdots \to S_m],\ [T_1 \to T_2 \to \cdots \to T_n] \mid}{\sum_{r=0}^{r=n-1} (n-r) \mid\mid [S_1 \to S_2 \to \cdots \to S_m],\ SqcComb([T_1 \to T_2 \to \cdots \to T_n],\ r) \mid\mid}$$

DSRM 類似度を計算するアルゴリズム

```
1  double[] SimDSRM( set_of_method_structure M,
2                    sequence [T1→T2→...→Tn] ) {
3    double Sim[M.length];
4    int Nume;
5    int Deno;
6    for ( int j=0;  j < M.length;  j++ ) {
7      Nume= Count( getMethodStructure(j), [T1→T2→...→Tn], 0 );
8      Deno= 0;
9      for ( int r=1;  r < [T1→T2→...→Tn].length;  r++ ){
10       Deno= Deno + Count( getMethodStructure(j), [T1→T2→...→Tn], r );
11     }
12     if ( (Nume + Deno) == 0 ) { Sim[j]= -1; }
13     else { Sim[j]= (double) Nume / (double)( Nume + Deno ); }
14   }
15   Return Sim;
16 }
```

Count 関数のアルゴリズム

```
1  int Count( method_structure MS, sequence TN, int R){
2    statement[] S;
3    statement[][] DS;
4    statement[] SV;
5    int CT=0;
6    // Generate derived sequence replacing R statements with negations
7    DS= SqcComb( TN, R );
8    for ( each S[] ∈ MS){
9      for ( int j=1;  j<=MS.length–TN.length;  j++ ){
10       for ( each SV[] ∈ DS ){
11         for ( int k=1;  k<= TN.length–R; k++ ){
12           if ( S[j] = SV[k] for all k-th elements that are not negative )
13                 { CT= CT + (TN.length–R) };
14         }
15       }
16     }
17   }
18   Return CT;
19 }
```



検索条件: *if{ → IOException → }*

No	メソッド名	SDRM類似度			Sorensen-Dice指標		
		類似度	完全一致	部分一致	類似度	完全一致	部分一致
1	SSIServletExternalResolver::getAbsolu	0.857	6	1	0.857	6	1
2	InputBuffer::readByte()	0.750	3	1	0.750	3	1
4	WsOutbound::flush()	0.750	3	1	0.750	3	1
5	UpgradeAprProcessor::write()	0.750	3	1	0.750	3	1
6	SecureNioChannel::close()	0.667	6	3	0.667	6	3
7	Conversions::byteArrayToLong()	0.600	3	2	0.600	3	2
8	BioReceiver::start()	0	0	11	0.545	6	5
9	FileUtils::forceDelete(File file)	0	0	9	0.333	3	6
10	InputBuffer::skip(long n)	0.200	3	12	0.200	3	12
11	ClassParser::parse()	0.158	3	16	0.158	3	16
12	MemoryUserDatabase::save()	0.097	3	28	0.290	9	22
13	InternalAprInputBuffer::fill()	0	0	23	0.261	6	17
14	SecureNioChannel::read()	0	0	19	0.158	3	16
15	NioBlockingSelector::read()	0.103	3	26	0.103	3	26

検索精度の向上 (Table 2)

No	検索されたシーケンス	メソッド数 SDRM指標	メソッド数 Sorensen-Dice指標	精度 向上率
1	if{ → Log.debug	143	151	5.3%
2	if{ → Log.debug → }	141	151	6.6%
3	if{ → StringBuilder.append	102	143	28.7%
4	catch{ → Log.error	112	131	14.5%
5	catch{ → ExceptionUtils.handleThrowable	97	111	12.6%
6	if{ → IllegalArgumentException	107	133	19.5%
7	if{ → IllegalArgumentException → }	107	133	19.5%
8	catch{ → Log.error → }	87	128	32.0%
9	if{ → org.apache.juli.logging.Log.debug	70	73	4.1%
10	if{ → StringBuilder.append → }	66	141	53.2%
11	if{ → org.apache.juli.logging.Log.debug → }	68	71	4.2%
12	if{ → StringBuilder.append → StringBuilder.append	41	140	70.7%
13	if{ → IOException	71	89	20.2%
14	if{ → IOException → }	71	89	20.2%
15	catch{ → MBeanException	30	31	3.2%
16	catch{ → MBeanException → }	30	30	0.0%
17	if{ → StringBuilder.append → StringBuilder.append → }	37	135	72.6%
18	catch{ → Log.error → } → }	66	116	43.1%
			平 均	23.9%

結　論

- 頻出コードをマイニングする機能を開発した.
- シーケンスの類似度を定義した.
- Apache-Tomcat 7 のソースコードに適用した結果、平均 23.9% の検索精度の向上を確認した.

今後の方針

- Java言語固有の機能を反映した構造情報の抽出機能を開発する.
- 本手法の有効性を確認するため、さらに多くのJavaソースコードで実験する.

ご清聴ありがとうございました