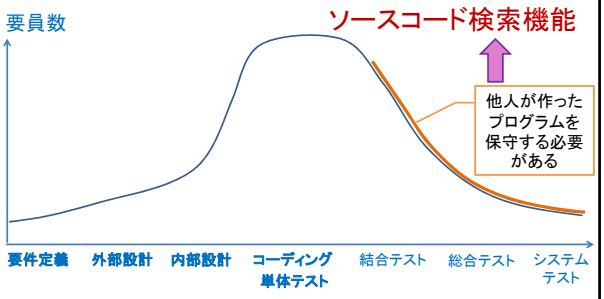
ソースコード検索のための シーケンスに基づく類似度について

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 (1/4)

情報システム開発のイメージ



1. 研究の背景と概要 (2/4)

実務面での応用

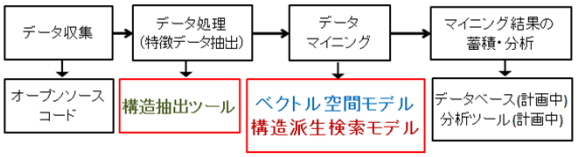
①バグあるいは脆弱なコードを探す
②高品質なコードを探す
③プログラム理解を支援する
④ソースコードを診断する

1. 研究の背景と概要 (3/4)

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、局所的な検索の留まる
③ 構造に基づく検索 ― 本研究のアプローチ
–文法に沿った検索ができるが、処理が複雑
④ 学習モデルによる比較
–修正すべきソースコード群が共通する特徴を有することが前提

1. 研究の背景と概要 (4/4)



- 構造抽出ツール --- C言語で実装
- ベクトル空間検索ツール --- VB言語で実装
- **構造派生検索ツール** --- VB言語で実装

目　次

2. 構造抽出ツールの主な機能(1/3)

対象はJava言語

(1) クラス名，メソッド名と引数

① クラス名::メソッド名(引数)
② クラス名:匿名クラス名:メソッド名(引数)

オブジェクト型

```
interface IntF{
  void printHello(int x);
}
class HelloAno {
  public static void main(String[] args) {
    IntF object1 = new IntF() {
      public void printHello(int x) {
        System.out.println("匿名クラス= "+ x);
      }
    };
    object1.printHello(767);
  }
}
```

匿名クラスの例

2. 構造抽出ツールの主な機能(2/3)

(2) Javaの制御文

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

2. 構造抽出ツールの主な機能(3/3)

(3) メソッドの中で呼び出しているメソッド名

変数名.メソッド名
⇒ クラス名.メソッド名
または
データ型.メソッド名

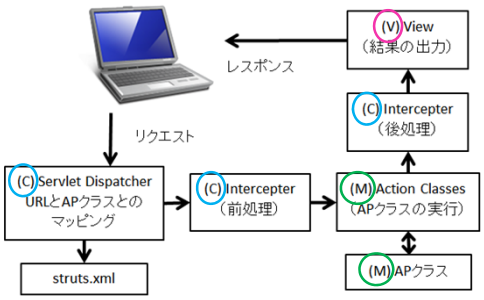
別物として扱う

変数名1.add (<String>) ⇒ **LinkedList <String>.**add(<String>)

変数名2.add (<String>) ⇒ **List<String>.**Add (<String>)

Web APフレームワーク Struts 2

- 2007年にリリースされた Webフレームワーク
- MVCモデルに準拠している



主なメトリックス

メトリックス	指標値
ファイル数	368個
クラス数	414個
メソッド数	2,677個
宣言文と実行文の行数	21,543行
コメント行数	17,954行
ソース総行数	46,100行
最大ネスト数:	8
最大循環的複雑度	55

制御文とメソッド名の構造の抽出結果

```
Settings::static Settings getDefaultInstance()
#   8  3 #
{
    if{
    DefaultSettings
        try{
        get
            if{
                try{
                (Settings) ObjectFactory.getObjectFactory()
                }
                catch{
                Logger.error
                }
            }
        }
        catch{
        }
    }
}
```

最大ネスト数8のメソッド

```
PrefixBasedActionMapper::
    ActionMapping getMapping(HttpServletRequest request,ConfigurationManager configManager)
#       1     0   38
{
getUr
    for{
    Map<String,ActionMapper> get
        if{
        ActionMapper.getMapping
            if{
            Log.debug
            }
            if{
                if{
                Log.debug
                }
                for{
                mappingParameterEntry
                    if{
                    Log.debug
                    }
                    else_if{
                    Log.debug
                    }
                    else_if{
                    Log.debug
                    }
                    else{
                    Log.debug
                    }
                }
            }
        }
        else_if{
        Log.debug
        }
    }
    if{
    Log.debug
    }
}
```

目　次

1. 研究の背景と概要
2. 構造抽出ツールの主な機能
3. ベクトル空間モデルと類似度
4. 構造派生検索モデルと類似度
5. 構造派生検索モデルによる検索結果
6. まとめと今後の研究方針

類似検索機能の比較方針

- ソースコードは，プログラミング言語としての文法に準拠した文書
- 自然言語で記述された一般的な文書との違いを比較する

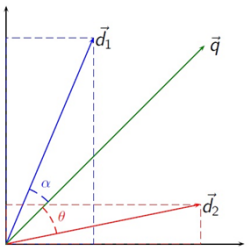
一般的な文書　⇒　ベクトル空間モデル
ソースコード　⇒　構造派生検索モデル

ベクトル空間モデルの概要

① 文書と検索条件を単語のベクトルとして表現する

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{N,j})$$

$$q = (w_{1,q}, w_{2,q}, \ldots, w_{N,q})$$



② 文書と検索条件の類似度を2つのベクトルの Cos で計算する

$$\text{Similarity}(\boldsymbol{d_j}, \boldsymbol{q}) = \cos(\boldsymbol{d_j}, \boldsymbol{q}) = \frac{\sum_{i=1}^{N} w_{i,j} * w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} * \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

VSMによる類似検索の結果

catch句とLogger.errorメソッドを含むソースの検索結果

（28から41までは省略）

行番	メソッド名	ベクトル空間モデル 類似度	構造派生検索モデル 類似度	完全一致数	部分一致数	コード行数
1	BaseTemplateEngine::void tryToLoadPropertiesFromStream()	0.577	1.000	2	0	9
2	FilterDispatcherCompatWeblogic61::void setFilterConfig()	0.577	1.000	2	0	9
3	ClosingUtilBean::boolean start()	0.535	1.000	2	0	11
4	FreemarkerManager::synchronized Configuration getConfiguration()	0.535	1.000	2	0	12
5	Settings::static Settings getDefaultInstance()	0.530	0.667	2	1	17
6	Anchor::boolean end()	0.500	1.000	2	0	13
7	Submit::boolean end()	0.500	1.000	2	0	12
8	FreemarkerManager::void loadSettings()	0.477	0.800	4	1	37
9	Bean::boolean start()	0.471	1.000	2	0	14
10	DefaultSettings::DefaultSettings()	0.471	0.500	2	2	24
11	FreemarkerManager::TemplateLoader createTemplateLoader()	0.471	1.000	2	0	12
12	ClassLoaderUtils::static Class loadClass()	0.452	0.000	0	3	15
13	FilterDispatcher::HttpServletRequest prepareDispatcherAndWrapRequest()	0.447	1.000	2	0	17
14	VelocityManager::VelocityEngine newVelocityEngine()	0.447	0.000	0	2	18
15	FastByteArrayOutputStream::void writeTo()	0.426	0.000	0	2	11
16	AnnotationValidationInterceptor::Method getActionMethod()	0.392	0.000	0	2	14
17	FastByteArrayOutputStream::void writeToFile()	0.392	0.000	0	2	15
18	InitOperations::void initLogging()	0.369	0.000	0	3	19
19	DefaultStaticContentLoader::void initLogging()	0.364	0.000	0	3	20
20	FilterDispatcher::void initLogging()	0.364	0.000	0	3	20
21	StrutsObjectFactory::Interceptor buildInterceptor()	0.362	0.000	0	1	38
22	PrepareOperations::HttpServletRequest wrapRequest()	0.354	0.000	0	2	10
23	TagModel::Map unwrapParameters()	0.354	1.000	2	0	22
24	JakartaMultiPartRequest::File[] getFile()	0.333	0.000	0	2	21
25	BackgroundProcess::void run()	0.316	0.000	0	1	1
26	Text::boolean end()	0.309	1.000	2	0	1
27	JakartaMultiPartRequest::void parse()	0.289	0.000	0	1	1

28～41は省略

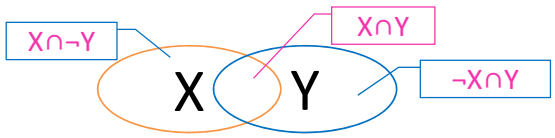
目　次

構造派生検索モデルと類似度

Jaccard類似度（係数）

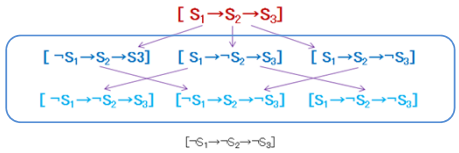
スイスの植物学者Paul Jaccard(1868-1944)に由来

$$Sim_{Jaccard}(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|X \cap Y|}{|X \cap Y| + |X \cap \neg Y| + |\neg X \cap Y|}$$



N=3 の場合

$$類似度 = \frac{|[S_1 \rightarrow S_2 \rightarrow S_3]|}{\begin{array}{l}|[S_1 \rightarrow S_2 \rightarrow S_3]| + \\ |[\neg S_1 \rightarrow S_2 \rightarrow S_3]| + |[S_1 \rightarrow \neg S_2 \rightarrow S_3]| + |[S_1 \rightarrow S_2 \rightarrow \neg S_3]| + \\ |[\neg S_1 \rightarrow \neg S_2 \rightarrow S_3]| + |[\neg S_1 \rightarrow S_2 \rightarrow \neg S_3]| + |[S_1 \rightarrow \neg S_2 \rightarrow \neg S_3]|\end{array}}$$



（1）最大が **1.0**　最小が **0.0**

（2）「検索条件を満たす個数」に比例した類似度

類似度計算法

```
Input:  set_of_structure M;
Input:  sequence [S1→S2→...→Sn];
Output: Sim[M.length];
 // Declare array Sim[].
double Sim[M.length];
 // Compute Similarity for each method in M.
int Nume;  int Deno;
for ( int j=0; j < M.length; j++ ) {
   Nume= Count(getMethodStructure(j), [S1→S2→...→Sn], 0);
   Deno= 0;
   for ( int r=1; r < [S1→S2→...→Sn].length; r++ ){
      Deno= Deno
            + Count(getMethodStructure(j), [S1→S2→...→Sn], r);
   }
  Sim[j]= (double) Nume / (double)( Nume + Deno );
}
```

Count(m, $[S_1 \rightarrow S_2 \rightarrow ... \rightarrow S_n]$, r)

構造情報 mにおいてシーケンス $[S_1 \rightarrow S_2 \rightarrow ... \rightarrow S_n]$ から r 個の要素を否定形で置き換えたシーケンスに一致する部分構造情報の数を返す関数

getMethodStructure(j)

検索対象とするメソッドの構造情報リストの j 番目のメソッドの構造情報を取り出す関数

目　次

構造派生検索モデルの検索結果

検索条件： [catch{ → Logger.error]

行番	メソッド名	ベクトル空間モデル 類似度	構造派生検索モデル 類似度	完全一致数	部分一致数	コード行数
1	BaseTemplateEngine::void tryToLoadPropertiesFromStream()	0.577	1.000	2	0	9
2	FilterDispatcherCompatWebloglo01::void setFilterConfig()	0.577	1.000	2	0	9
3	ClosingLIBean::boolean start()	0.535	1.000	2	0	11
4	FreemarkerManager::synchronized Configuration getConfiguration()	0.535	1.000	2	0	12
5	Settings::static Settings getDefaultInstance()	0.530	0.667	2	1	17
6	Anchor::boolean end()	0.500	1.000	2	0	13
7	Submit::boolean end()	0.500	1.000	2	0	13
8	FreemarkerManager::void loadSettings()	0.477	0.800	4	1	37
9	Bean::boolean start()	0.471	1.000	2	0	14
10	DefaultSettings::DefaultSettings()	0.471	0.500	2	2	24
11	FreemarkerManager::TemplateLoader createTemplateLoader()	0.471	1.000	2	0	12
12	ClassLoaderUtils::static Class loadClass()	0.452	0.000	0	3	15
13	FilterDispatcher::HttpServletRequest prepareDispatcherAndWrapRequest()	0.447	1.000	2	0	17
14	VelocityManager::VelocityEngine newVelocityEngine()	0.447	0.000	0	2	18
15	FastByteArrayOutputStream::void writeTo()	0.426	0.000	0	2	11
16	AnnotationValidationInterceptor::Method getActionMethod()	0.392	0.000	0	2	11
17	FastByteArrayOutputStream::void writeToFile()	0.392	0.000	0	2	11
18	InitOperations::void initLogging()	0.369	0.000	0	2	11
19	DefaultStaticContentLoader::void initLogging()	0.364	0.000	0	2	11
20	FilterDispatcher::void initLogging()	0.364	0.000	0	2	11
21	StrutsObjectFactory::Interceptor buildInterceptor()	0.362	0.000	0	2	21
22	PrepareOperations::HttpServletRequest wrapRequest()	0.354	0.000	0	3	15
23	TagModel::Map unwrapParameters()	0.354	1.000	2	0	23
24	JakartaMultiPartRequest::File[] getFile()	0.333	0.000	0	2	21
25	BackgroundProcess::Thread void run()	0.316	0.000	0	1	10
26	Text::boolean end()	0.309	1.000	2	0	23
27	JakartaMultiPartRequest::void parse()	0.289	0.000	0	1	10
28	Component::boolean end()	0.289	0.000	0	1	12
29	BaseTemplateEngine::InputStream tryReadingPropertyFileFromFileSystem()	0.289	0.000	0	1	11
30	JspTemplateEngine::void renderTemplate()	0.258	0.000	0	2	34
31	VelocityResult::void doExecute()	0.250	1.000	2	0	36
32	Template::List<Template> getPossibleTemplates()	0.236	0.000	0	1	14
33	PropertiesSettings::PropertiesSettings()	0.236	0.000	0	2	28
34	MultiPartRequestWrapper::MultiPartRequestWrapper()	0.236	0.000	0	1	13
35	Date::boolean end()	0.229	1.000	4	0	62
36	DelegatingSettings::Iterator listImpl()	0.224	0.000	0	1	21
37	IteratorTag::int doAfterBody()	0.224	0.000	0	1	15
38	FilterDispatcher::void doFilter()	0.218	1.000	2	0	42
39	VelocityTemplateEngine::void renderTemplate()	0.213	0.000	0	2	39
40	Component::Object findValue()	0.204	0.000	0	1	17
41	ClassSelectionProvider::void register()	0.204	1.000	2	0	54

DSRの貢献度＝
(41-18)／41＝56.1%

検索条件と貢献度（表3）

行番	検索パターン	メソッド数 (VSM)	メソッド数 (Tf-idf VSM)	メソッド数 (DSR)	貢献度 (対VSM)	貢献度 (対Tf-idf VSM)
1	if{ → addParameter	27	25	21	22.2%	16.0%
2	if{ → Logger.debug	72	61	31	56.9%	49.2%
3	if{ → Logger.warn	45	32	13	71.1%	59.4%
4	while{ → StringTokenizer.nextToken	16	8	6	62.5%	25.0%
5	while{ → Enumeration.nextElement	13	11	5	61.5%	54.5%
6	while{ → Iterator.next	28	21	4	85.7%	81.0%
7	for{ → iterator.next	21	13	9	57.1%	30.8%
8	for{ → List<String>.add	5	6	4	20.0%	33.3%
9	for{ → StringBuilder.append	10	12	3	70.0%	75.0%
10	catch{ → Logger.error	41	43	18	56.1%	58.1%
11	if{ → addParameter → }	28	26	21	25.0%	19.2%
12	if{ → Logger.warn → }	49	21	3	93.9%	85.7%
13	if{ → Logger.debug → }	98	38	26	73.5%	31.6%
14	if{ → } → }	154	136	9	94.2%	93.4%
15	if{ → } → if{	206	175	22	89.3%	87.4%
16	if{ → } → else{	192	151	22	88.5%	85.4%
17	for{ → if{ → }	30	23	9	70.0%	60.9%
18	if{ → addParameter → } → else{	14	19	4	71.4%	78.9%
19	if{ → Logger.warn → } → }	58	40	24	58.6%	40.0%
20	if{ → } → } → }	247	197	18	92.7%	90.9%
21	for{ → if{ → } → }	13	23	9	30.8%	60.9%

性能測定結果

DSRはVSMよりも検索時間が短い

行番	検索パターン	VSM計算	Tf-idf VSM Tf-idf生成	Tf-idf VSM VSM計算	DSR
1	if{ → addParameter	35553	56466	35618	546
2	if{ → Logger.debug	36988	57270	35907	499
3	if{ → Logger.warn	36005	56903	36458	500
4	while{ → StringTokenizer.nextToken	36145	56943	35523	515
5	while{ → Enumeration.nextElement	36161	58495	36444	515
6	while{ → Iterator.next	35943	57745	36351	500
7	for{ → iterator.next	36083	56966	35819	515
8	for{ → List<String>.add	36551	55968	36388	515
9	for{ → StringBuilder.append	35880	57916	37745	514
10	catch{ → Logger.error	36482	56044	35322	499
11	if{ → addParameter → }	36645	57683	36615	1124
12	if{ → Logger.warn → }	37222	57137	36506	1292
13	if{ → Logger.debug → }	36551	57511	35524	1123
14	if{ → } → }	36474	56981	36462	1388
15	if{ → } → if{	36364	56403	36164	1388
16	if{ → } → else{	36582	57668	36302	1404
17	for{ → if{ → }	36286	56887	36520	1466
18	if{ → addParameter → } → else{	35662	56622	35069	2824
19	if{ → Logger.warn → } → }	36489	56606	36411	2839
20	if{ → } → } → }	35974	56934	35864	2908
21	for{ → if{ → } → }	34211	57291	35257	3034

目　次

まとめと今後の研究方針

- ソースコードは文のシーケンスが意味を持つことから、文のシーケンスを検索条件とする検索モデルを提案し、Struts 2 Coreソースへの適用結果を示した
- ベクトル検索モデルと比較して、機能、性能面での問題は見当たらなかった
- 今後は、
  ① Jaccard係数を使った類似検索と比較する
  ②オープンソースコードを使った実験を行う

ご清聴ありがとうございました