RO-014

既存ソフトウェア部品を用いたソフトウェア開発における ソースコード理解支援ツール

Support Tool for Understanding Source Code on Development with Existing Software Components

青山 裕介† 黒岩 丈瑠†‡ 久代 紀之†
Yusuke Aoyama Takeru Kuroiwa Noriyuki Kushiro

1. まえがき

既存ソフトウェアを再利用するためには，再利用を行う既存ソフトウェアの構造を理解しなければならない．既存ソフトウェアの構造を理解するためには既存ソースコードの分析が必要となる．ソースコードには，分岐や例外といった制御構造が含まれるが，これら構造が複雑化した場合には，if・else といった分岐の対応や，ある分岐条件のもとで実行される処理の範囲などを把握することが難しくなるため，構造の概略を書き下すなど，ソースコード外に補助情報を作成し，この補助情報を参照しながらソースコードを読み進めることが行われる．

本研究では，制御構造を可視化することで，ソースコード理解の障害となる複雑な制御構造の把握を支援するツールを，Java SE 7 の Java 言語[1]を対象とし，統合開発環境である Eclipse[2]上のプラグインとして実装した．

以降では，2 節でツールの可視化図式，実装方法について説明し，3 節でツールの評価実験の内容を，4 節で実験の結果を述べ，5 節で結果を考察し，6 節で関連研究を紹介する．

2. 方法

本研究では，複雑な制御構造を持つメソッドを読む際に，ツールによってその制御構造の概要を表すことで，メソッドの制御構造の理解を支援すること，また，メソッド中に出現するクラスや，クラス同士の参照関係を示し，メソッド中に出現するクラスや，クラス間の関係の発見を支援することを目指した．項 2.1 ではツールの可視化図式について説明し，項 2.2 ではツールの実装について説明する．

2.1 可視化図式について

本研究では，Java の制御構造を表す構文を，表 1 のように分岐・繰り返し・例外の 3 種類に分け，それぞれにシンボルとなるマークと，実行経路・範囲を表す矢印にて囲まれた図によって 制御構造の可視化を行った（図 1，図 2，図 3，図 4，図 5，図 6，図 7）．なお，制御構造はメソッド中に現れるため可視化図式はメソッドごとに生成した．分岐・繰り返しについては，条件に一致する経路とそうでない経路を並べて配置することによって，処理の違いを比較しながら読み進めることが出来る．また，class name を四角で囲む図をダブルクリックすることで，ソースコードの該当する位置に移動することができ，可視化図式によって，大まかな構造把握を行いながら，注目したい部分についてソースコードによって読み進める，といったことが可能になる．さらに，色付きで class name を囲み，参照関係にあるクラス同士を示すことも行った（図 8）．例えば，図 8 中では緑色の注目クラス（demo.D）を中心として，紫色のクラス（demo.A）は注目クラスと相互に参照関係にあることを表し，青色のクラス（demo.B）は注目クラスから参照されていることを表している．注目クラスは class name を囲む図をクリックすることで変更することが出来，参照関係にあるクラスを示すことで，制御構造だけでなく，開発者にとってブラックボックスとなっている 既存ソフトウェアのクラス間の関係把握も支援することを目指した．また，複雑な制御構造を可視化することで，可視化図式が画面内に収まらず全体像が把握できなくなることを防ぐために，可視化図式表示時には，Outline 部分に可視化図式のミニマップ表示を行った（図 9）．

表 1 可視化を行った構文要素

構文	制御構造
if 文	分岐
switch 文	分岐
while 文	繰り返し
do 文	繰り返し
for 文	繰り返し
拡張 for 文	繰り返し
try 文	例外

2.2 ツールの実装について

本研究では，Java SE 7 の Java 言語[1]を対象とし，Eclipse プラグインとして実装を行った．ツールは以下の手順で動作する．

1. 可視化を行いたいメソッドを選択し，ツールを起動する
2. ツールは選択されたメソッドのソースコードを取得する
3. 取得したソースコードを Eclipse JDT[3]を用いて AST[4]に変換する
4. 変換された AST を探索し，ソースコード中に変数・メソッド呼び出し・メンバ宣言を参照関係として記録する
5. AST 構造を GEF（Graphical Editing Framework）を用いて可視化図式に変換し，表示する

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

```
public void demo(boolean flag){
    if (flag)
        System.out.println("if");
    else
        System.out.println("else");
}
```

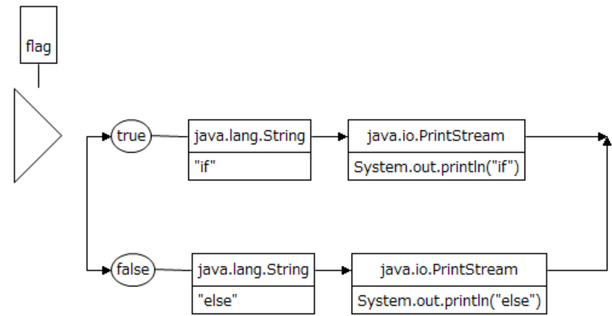

図 1 if 可視化例

```
public void demo(AccessMode mode){
    switch (mode){
    case EXECUTE:
        System.out.println("execute");
        break;
    case READ:
        System.out.println("read");
    default:
        System.out.println("default(write)");
    }
}
```

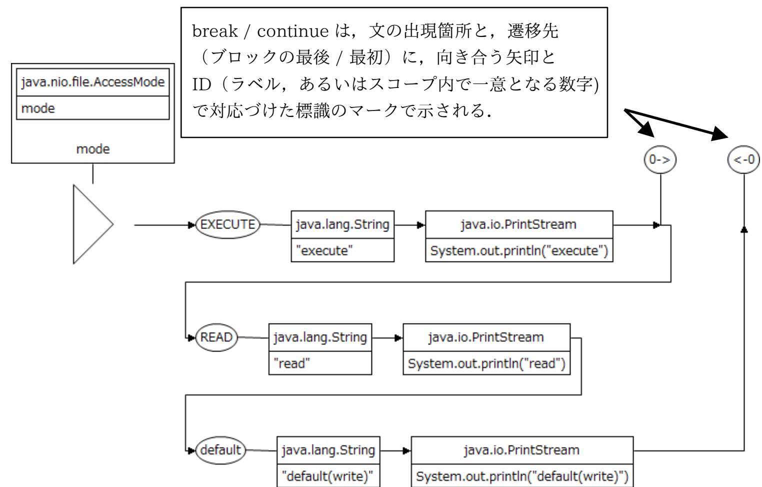

図 2 switch 可視化例

```
public void demo(int num){
    while(num-- > 0)
        System.out.println("demo");
}
```

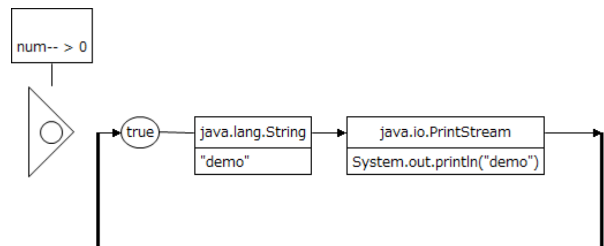

図 3 while 可視化例

```
public void demo(List<String> list){
    for(Iterator<String> iter = list.iterator();
            iter.hasNext(); ) {
        System.out.println(iter.next());
    }
}
```

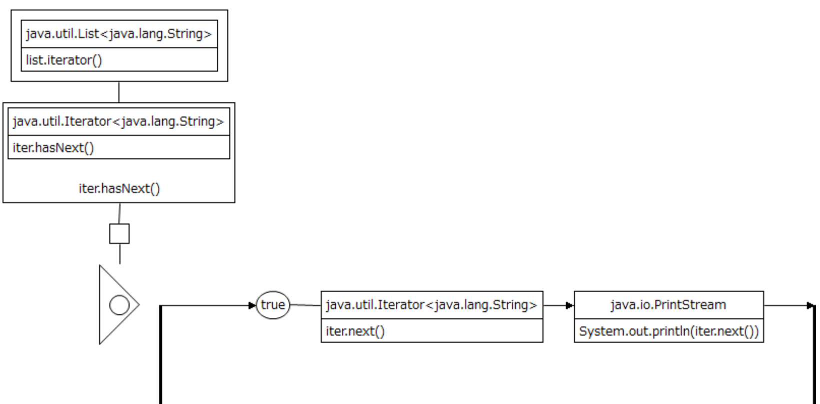

図 4 for 可視化例

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

```
public void demo(List<String> list){
    for(String str : list) {
        System.out.println(str);
    }
}
```

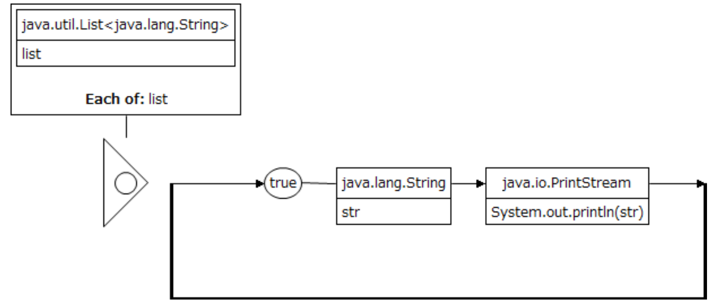

図 5 拡張 for 可視化例

```
public void demo(int num){
    int count = 0;
    do {
        System.out.println("demo");
    } while(count++ < num);
}
```

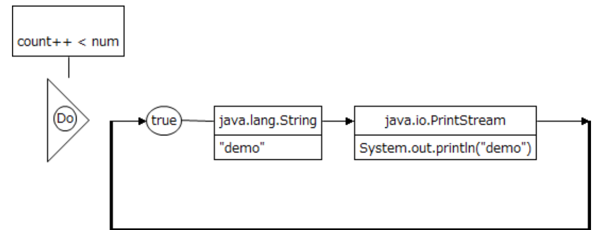

図 6 do 可視化例

```
public void demo(int left, int right){
    try{
        System.out.print(left / right);
    } catch(ArithmeticException e){
        e.printStackTrace();
    } finally{
        System.out.println();
    }
}
```

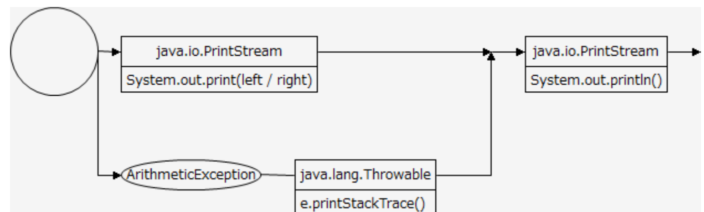

図 7 try 可視化例

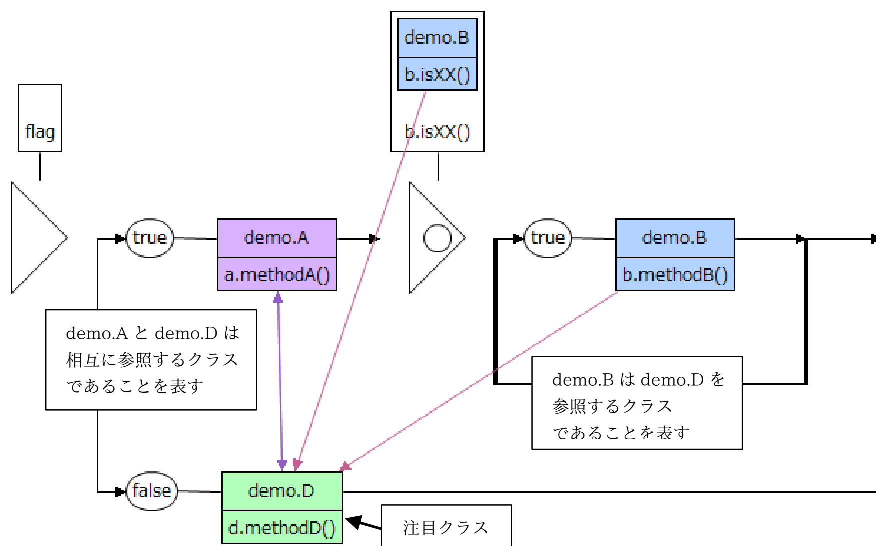

図 8 クラスの参照関係可視化例

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

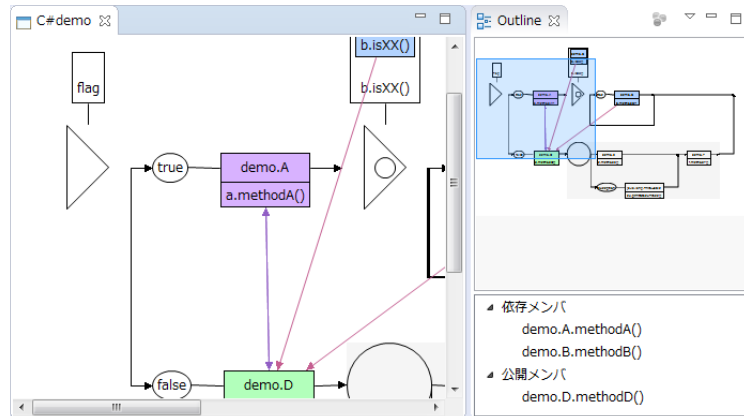

図 9 可視化図式のミニマップ表示

ツールの起動方法には以下の 2 種類がある．

1. Explorer や Outline などから可視化を行いたいメソッドを選択し，右クリックメニューからツール起動の項目を選択する
2. ソースコード中の可視化を行いたい箇所にキャレットを置き，右クリックメニューからツール起動の項目を選択する

特に，2 では，可視化図式を表示する際に，キャレットを置いた周辺と対応する箇所まで 画面をスクロールした状態で可視化図式が表示されるようになっている．

また，可視化図式とソースコード間の双方向の移動を可能とするために，項 2.1 で述べたように可視化図式からもソースコードに移動する事ができるようにした．

3. 実験

設計意図通りにツールが用いられることを確認するために，被験者に既存プログラムに対して，項 3.1 で示すような追加仕様を実現する処理を記述させ，ツールの有無による作業の変化を観察した．被験者の作業プロセスを明らかにするために，作業中に被験者が考えていることを発話してもらうプロトコル分析を行い，発話と作業中の画面を記録し，観察された作業を表 2 のように分類した．また，普段の開発作業のプロセスが記録できるように，ツールを使用しない場合には被験者が普段 Java による開発に用いているエディタを使用させた．被験者は表 3 の 4 名に対して表 4 のように課題を割り当てた．

3.1 課題について

被験者には，Java SE Development Kit 7u65 Demos and Samples[5]に含まれるプログラム Font2DTest と SampleTree を元に，項 3.1.1，項 3.1.2 のような 追加仕様を実現するようなプログラムを記述させた．

3.1.1 Font2DTest

既存プログラムでは，起動するとアルファベットが特定のフォントで表形式に並んだウィンドウが開く．描画されている文字をクリックすると，クリックした文字が拡大表示される機能がある．追加仕様として，クリックした時の文字列を標準出力に書き出すような処理を追加させた．この追加仕様を実現するためには，クリックした際の文字を取得する方法を知る必要がある．ソースコード中でクリックした文字は，内部では int 型の表現されているため，これを文字列に変換する方法を調べなければならない．さらに，既存プログラムにはアルファベットの描画方法を変更する機能があり，機能毎に文字列への変換方法が異なっているため，被験者は描画方法ごとの分岐や，既存プログラムにある，追加仕様には関係しないその他のモードごとの分岐を読み解き，適切な箇所に追加処理を記述しなければならない．

表 2 作業分類

作業名	作業内容
Searching class or method	ソースコードをスクロールする，検索機能によってクラス，あるいはメソッドを探している
Reading method	メソッドを読んでいる
Editing	ソースコードを編集している
Using tool	ツールを利用している
Running program	プログラムを実行し，動作や実行時エラーを確認している
Others	課題やツールの利用方法についての質問が行われている，Java の標準ライブラリについて，Javadoc を読んでいるか，あるいは質問している等，上記の作業に該当しない作業が行われている

3.1.2 SampleTree

既存のプログラムでは，起動すると，コンピュータにインストールされているフォントの中からランダムに数種類のフォントが選ばれ，選ばれたフォント名が各ノードで配置されたツリーを描画したウィンドウが開く．ツリー中のノードをクリックすると子ノードが一定数追加される．追加仕様として，この子ノードの個数を制御できるようなテキストフィールド追加させた．

この追加仕様を実現するためには，追加処理が行われている箇所を特定する必要がある．ノード追加処理を行って

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

表 3 被験者情報

被験者	所属	開発経験	普段 Java 開発に用いるエディタ
A	電機メーカ	10 年	Eclipse
B	電機メーカ	20 年	サクラエディタ
C	大学 4 年生	1 年	Eclipse
D	大学 4 年生	1 年	NetBeans

表 4 課題の割り当て

被験者	1 回目の課題	2 回目の課題
A	Font2DTest（ツール無）	SampleTree（ツール有）
B	Font2DTest（ツール有）	SampleTree（ツール無）
C	SampleTree（ツール無）	Font2DTest（ツール有）
D	SampleTree（ツール有）	Font2DTest（ツール無）

いる部分ではフォントごとに表示可能かどうかの分岐や，描画色の指定などの分岐があり，追加処理を見つけるためには，被験者はこれら分岐を読み解き，適切な箇所に追加処理を記述しなければならない．

課題として与える 2 種のプログラムについて，上記のような追加仕様を実現するためには，どこに追加処理を記述すればよいかを探すためにソースコードを読み解く作業が必要となる．特に上記仕様を満たすためには種々の分岐構造を読み解く作業が必要となり，目的通りのツールが作成できているならば，このような分岐などの構造を把握するためにツールが用いられることが期待できる．

4. 実験結果

課題中に観察された作業を表 2 のように分類し，作業間の遷移をグラフ化した（図 12，図 13，図 14，図 15）．図中破線のものはツールの有無に関係なく共通して存在する作業・遷移であり，太線のものはツール有の場合のみに存在する作業・遷移であり，その他の線はツール無の場合のみに存在する作業・遷移である．被験者の作業時間は図 11 である．

5. 考察

本研究では，節 2 前半で述べた設計意図に基づいて実装を行った．実際の使用ケースとしては，以下を想定している．

- **制御構造の可視化機能**
  構造が複雑なメソッドを読む前，あるいは最中に，制御構造の可視化図式をリファレンスとし，構造の理解や，メソッド中の読みたい場所の特定に用いられる．
- **クラス間の参照関係の可視化機能**
  クラス間の参照関係などを元に，読み進めるべきクラスを発見するために用いられる．

したがって，表 2 の分類においては，ツールを使用する作業「Using tool」は，メソッドの制御構造理解のために，メソッドを読む作業である「Reading method」との間に，

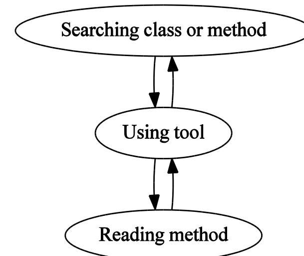

図 10 期待する遷移

また，読み進めるクラスを見つけるために，クラスを探す作業である「Searching class or method」との間に，図 10 のような遷移を持つことが期待される．以下ではこれについて作業の遷移を表すグラフ（図 12，図 13，図 14，図 15） 及び，実験中でのツールの使用シーン（表 5）から確認する．

被験者 A は，図 12 より，ツールはメソッドを探す作業，メソッドを読む作業の間のどちらでも用いられることがわかる．これら作業間の遷移について，表 5 を参照すると，

- **「Searching class or method」との間**
  想定した使用シーンとは異なるが，ツールは，メソッド中の読むべき箇所を特定するために用いられており，ツールの設計意図通りの使用がなされている．
- **「Reading method」との間**
  メソッドの制御構造の把握のためにツールは用いられ，ツールの設計意図と合っている．

また，実験後にどのようなときにツールを使用したのか，という質問に対しても，ループの確認などを目的として用いた，という回答が得られ，ツールに対する認識も設計意図通りであった．

被験者 B は，図 13 より，ツールはメソッドを探す作業，メソッドを読む作業の間のどちらでも用いられることがわかる．これら作業間の遷移について，表 5 を参照すると，

- **「Searching class or method」との間**
  設計意図とは異なる使用であるが，ツールの可視化内容についての誤解により発生した遷移であり，実験中に誤解である旨と，ツールの機能について再度説明したところ実験中再び起こることはなかった．
- **「Reading method」との間**
  ツールはメソッドの制御構造の把握や，読みたい場所の特定のために用いられており，これは設計意図通りの使用である．

また，実験後，どのようなときにツールを使用したのかを尋ねたところ，

- if 文があって，続けてどのような処理をしている，といった大きな流れが見たかったから
- 関数や演算がたくさんでてくるが，ソースコードを一つ一つ追うと時間がかかるのに対し，図だと大きく処理を見ていけるから

という回答を得た．これらから，被験者 B はメソッド中の大まかな流れを把握するためという，設計意図に合う認識がなされていたとわかる．

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

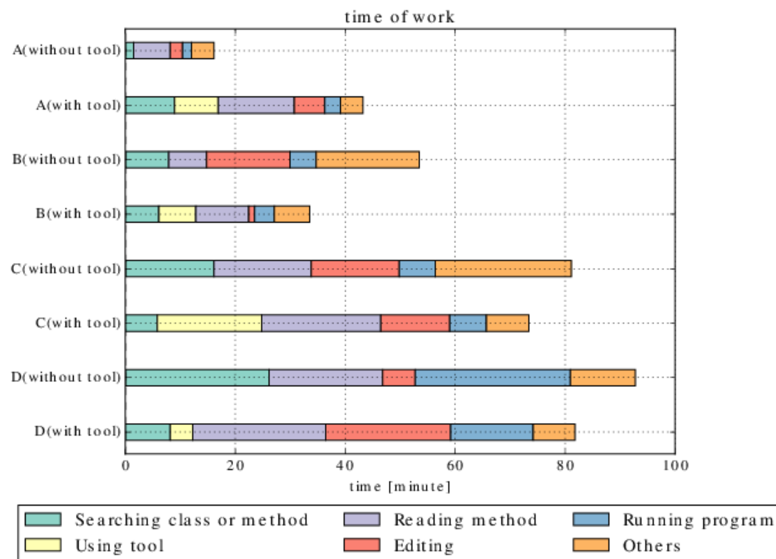

図 11 作業時間

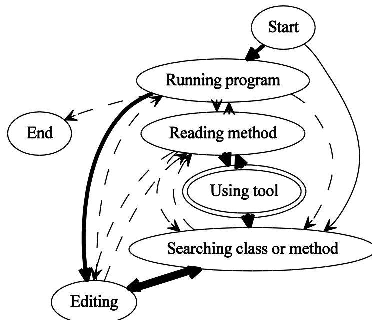

図 12 作業遷移（被験者 A）

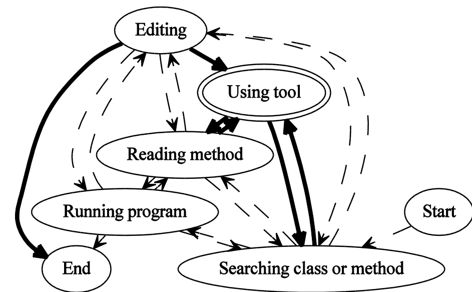

図 14 作業遷移（被験者 C）

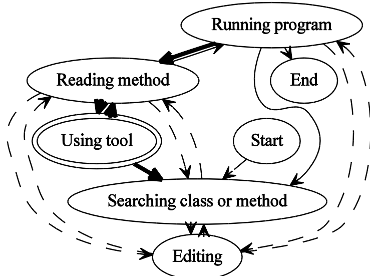

図 13 作業遷移（被験者 B）

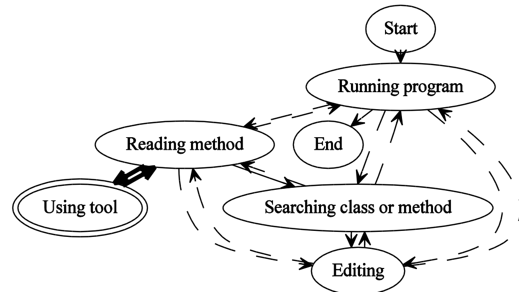

図 15 作業遷移（被験者 D）

被験者 C は，図 14 より，ツールはメソッドを探す作業，メソッドを読む作業の間のどちらでも用いられ，加えてソースコードを編集する作業間でも使用されていることがわかる．これら作業間の遷移について，表 5 を参照すると，

- **「Reading method」，「Editing」との間**
  ツールの制御構造の可視化図式をリファレンスとしながら，ソースコードを読み進めていることが見られ，これは設計意図通りの使用である．
- **「Searching class or method」との間**
  メソッドがどのような処理を行っているのか，どのようなクラスが出現するのかを可視化図式によって確認する，ということが行われており，これは設計意図通りの使用である．

また，実験後にどのような時にツールを使ったか，という質問を行ったところ，

- 細かいところではなく，構造が見たいときにツールを使った
- 知らないプログラムを読む際にプログラムの動きを書くが，ツールも同様のことを表現していると考え，読み進めるメソッドがどういう動きをしているのかを追う目的で使った

という回答が得られた．ここから，被験者 C は，メソッドの大まかな流れを把握するために用いたことが確認され，ツールに対する認識についても設計意図通りのものであった．

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

表 5 実験中でのツールの使用ケース

	「Searching class or method」間	「Reading method」間	「Editing」間
被験者A	ツール中で被験者が可視化を行ったメソッド中にラジオボタン追加処理があると予測し，可視化図式中から探す．しかし，可視化を行ったメソッド中にはその処理はなく，見つからなかったために，ツールの使用をやめてソースコードに移動し，検索機能によって被験者が読みたい処理を行っているメソッドを探す，といったことが行われていた	メソッドを読む際にメソッド中に繰り返し処理があるか確認するため，また，GUI 部品に対する入力値の取得する方法を知るために，既存のソースコードではどのような処理になっているのかを確認する際にツールで可視化を行う，といったことが行われていた．	
被験者B	ツールの用い方に関する誤解から生じた．メソッドを読んでいる途中で，メソッド中の確認したいメソッドが有り，可視化を行おうと それについて今まで読んでいたメソッドについてツールを起動して可視化を行う．しかし，ツールが可視化するのは確認したいメソッドではなく，確認したいメソッドを呼び出している （今まで読んでいた）メソッドであることに気づき，ソースコードに戻り，検索機能で読みたいメソッドの宣言を探す，といったことが行われていた．	・「Reading method」から「Using tool」 ソースコード中のメソッドを眺め，メソッドの構造が複雑である旨の発言の後に ツールによる可視化を行って流れを追いかける，といったことが行われていた． ・「Using tool」から「Reading method」 可視化図式中からズームする文字列を特定する処理を探し，被験者が該当処理であろうと予想した処理について，ツールからソースコードへ移動し，移動した処理周辺を読む，といったことが行われていた．	
被験者C	・「Searching class or method」から「Using tool」 右記の通り，メソッドを読む際はまずツールによって流れを追うということを行っていたため，メソッドを探し，見つけたメソッドについてツールによる可視化が行われていた． ・「Using tool」から「Searching class or method」 注目するメソッドがどのような処理をしているのかを確認するためにツールにより可視化を行うが，期待する処理を指定なかったので別のメソッドを探す，ということが行われていた．	メソッドを読む際に，まずツールによって流れを追い，注目する箇所について可視化図式からソースコードへ移動してメソッドを読み，読んでいるメソッドについて課題の追加仕様のための編集を行った後，再度ツールによって流れを追いかける，といったことが行われていた	
被験者D		・「Reading method」から「Using tool」 メソッドを読んでいる際に，メソッドが何をしているのかわからなくなった，という旨の発言の後に，読んでいるメソッドをツールによって可視化して見る，といったことが行われていた． ・「Using tool」から「Reading method」 可視化図式中からクラスの図を選択して ソースコードに移動し，選択図式周辺の処理を読むといったことが行われていた．	

被験者 D は，図 15 より，ツールはメソッドを読む作業の間でのみ用いられていることがわかる．このような遷移について，表 5 を参照すると，

- 「Reading method」との間
  被験者にとって読解困難なメソッドの流れを把握するためや，ソースコード中の読みたい処理の発見のために用いられており，これは設計意図通りの使用である．

また，実験後にどういう時にツールを使用したのかという質問を行ったところ，メソッドがどういう流れで動いているのかという構造を見たかった時に用いた，という回答を得た．これらから，被験者 D は，ソースコードからは流れがわかりにくいメソッドについて，その構造を見るために，ツールを使用していることが確認でき，ツールに対する認識についても設計意図通りのものであった．

以上から，ツールはメソッドの制御構造の把握という設計意図に合う使用をなされていることが確認され，ソースコードの理解支援に寄与することを確認した．一方で，ツール中の可視化図式がクラス間の関係の発見に用いられることはなかった．これについて，実験後，なぜクラス間の関係発見に用いなかったかを尋ねたところ，どの被験者も参照関係の発見が必要なタスクはないという認識をしていたことがわかった．今回実験で用いた課題のプログラムは，Java SE 7 のデモ用のプログラムであったため，プログラム中で出現するクラスのほとんどが Java の標準ライブラリを用いたものであり，また，追加仕様についても，Java の標準ライブラリを用いて実現できたために，課題プログラム中のクラス間の関係について理解する必要がなかったことが原因と考える．

6. 関連研究

プログラム理解を支援する関連研究について紹介する．

[6]は，Java ソースコードを解析して，注釈付きのプログラム表示を行うシステムのプロトタイプについて述べられている．ソースコード上に型情報をルビによって付与し

Copyright © 2015 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

たり，注目箇所以外の情報を削ったりと，ソースコードを装飾することでプログラムの理解を支援が目指されている．プロトタイプとしては，ソースコードを，型情報などを含んだ XML に変換したものを，XSLT によってソースコードに装飾を行った HTML に変換し，表示するといったことが行われている．

[7]は，メソッド間の依存関係を解析し，解析した依存関係から再利用を行うソフトウェア部品の関連する一連の機能を収集し，ソフトウェア部品を理解するために必要な情報を提示する，といったことが行われている．提示される情報には，依存関係のツリー表示や，プログラム理解で重要な役割を持つと推測される部品がある．

[8]は，ソフトウェアの修正や変更を行う際に，対象となる部分を開発者に提示するツールの開発が行われている．対象箇所の抽出のために，変数の出現範囲や制御構造の内容を抽出するフィルタが定義されており，開発者はこれらフィルタを単独で用いる，あるいは組み合わせて絞り込みを行うことで必要な情報を取得する．

[9]は，Java のソースコードについて，コールグラフを可視化するものであり，メソッドの制御フローの調査を支援している．[9]のコールグラフは，ノードが実行される順序通りに表示されるように配置され，また，各ノードからの呼び出し関係を対話的に展開することで グラフの複雑化を抑える，などの工夫がなされている．

[10]は，Java プログラムについてエイリアス解析を行い，メソッドの入力として用いられている引数やフィールド，クラス変数について，その変数に対して読み込み，あるいは書き込みが行われているのか，といった利用情報やそれら変数を用いるメソッドのソースコード中での位置と言った情報を木構造で可視化することで，メソッドの入力データ調査を支援することが行われている．

7. むすび

本研究では，ソースコード理解の障害として，複雑な制御構造の把握の難しさを課題とし，制御構造の可視化によってソースコードの理解支援を行うツールの開発を行った．開発したツールが制御構造の把握支援という目的のために使用されることを確認するため，4 人の被験者に既存プログラムに対する機能追加という開発作業与え，その作業を観察した．その結果，ツールはソースコードの制御構造の把握という設計意図通りに使用され，ソースコードの理解支援に寄与することを確認した．

節 3 の実験の後に被験者にツールの改善点についてのアンケートを行った結果，可視化図式中の文字列検索ができないか，といった新機能に関する提案や，ツールを使っている際に分岐や繰り返しのシンボルが何を表すものかわからなくなったという表記の問題点の指摘を受けた．実験中，ソースコード中から目的とするメソッドや変数を探したり移動したりする際にエディタの検索機能を用いていることが確認されており，可視化図式中でもこのような操作が可能となるような機能を実装したい．また，可視化図式に関してシンボルが何を表すものかわからなくなった，という指摘に関して，本研究で分岐や繰り返しのシンボルとして用いている図形が，三角や丸といった単純図形であり，被験者にとって分岐や繰り返しを表すための図形として直感的でなかった事が考えられる．これら可視化の図式表現についても洗練していきたい．

参考文献

[1] Gosling James，Joy Bill，Steele Guy，Bracha Gilad，Buckley Alex，“The Java Language Specification，Java SE 7 Edition”，(2013)，

[2] Eclipse Foundation，“Eclipse”， https://eclipse.org/

[3] Eclipse Foundation，“Eclipse JDT”， https://eclipse.org/jdt/

[4] Aho V．Alfred，Sethi Ravi， Ullman D．Jeffrey，“*Compilers: Principles，Techniques，and Tools (2nd Edition)* (原田 賢一(訳)，コンパイラ [第 2 版] --原理・技法・ツール--，サイエンス社，2009)”，(2006)

[5] Oracle，“Java SE Development Kit 7u65 Demos and Samples”

[6] 田中 哲，“ソースコード理解支援ツール-JavaMarkup”，日本ソフトウェア科学会 SPA2002，(2002)

[7] 小堀 一雄，山本 哲男，松下 誠，井上 克郎，“メソッド間の依存関係を利用した再利用支援システムの実装”，電子情報通信学会技術研究報告．SS，ソフトウェアサイエンス，Vol.104，No.722 (2005)

[8] 新倉 諭，鈴木 正人，“ソースコード理解支援機能を持つ開発環境”，情報処理学会研究報告．ソフトウェア工学研究会報，Vol.2008，No.29，(2008)

[9] LaToza D．Thomas，Myers，A．Brad，“Visualizing call graphs”，Visual Languages and Human-Centric Computing (VL/HCC)，2011 IEEE Symposium on，(2011)

[10] 鹿島 悠，石尾 隆，井上 克郎，“エイリアス解析を用いたメソッドの入力データの利用法可視化ツール”，Vol.2012，(2012)

Copyright © 2015 by Information Processing Society of Japan and The Institute of Electronics, Information and Communication Engineers All rights reserved.