Vol.2020-SE-206 No.13
2020/11/27

オブジェクトの移動に関する協調動作の動的グラフによる可視化手法

本郷 亜季[1,a)]　新田 直也[1,b)]

概要：ソフトウェアの保守・再利用作業においてプログラムを理解するために，その動作を可視化することは有効である．しかしながらオブジェクト指向プログラムの動作には，さまざまな側面があり，既存の表現形式でそのすべての側面を可視化できるわけではない．本研究では，メッセージを介したオブジェクトの移動に関する協調動作に着目し，その協調動作の複雑性に対応するため，動的グラフを用いた可視化手法の構築を目指す．まず本稿では，対象とする複雑性を交替複雑度として定式化し，その指標を評価するための被験者実験を行った．その結果，交替複雑度と，被験者がプログラムを理解する上での困難さがよく一致していることを確認した．さらに，交替複雑度が高い協調動作に対応するため，本研究室で提案しているデルタ抽出と呼ばれる動的解析技術に基づいて可視化を行うツールのプロトタイプを開発した．今後，実際のソフトウェアにおける協調動作の理解を支援できるようプロトタイプを完成させていく予定である．

1. はじめに

オブジェクト指向プログラムの実行時には，さまざまなオブジェクトが生成され，その間でさまざまなメッセージが送受信される．また，オブジェクトはフィールドを介した参照によって結ばれ，全体として巨大なグラフを構成する．このようなプログラムの内部動作を，さまざまな側面から可視化する表現手法が存在している．例えばシーケンス図では，オブジェクト間のメッセージのやり取りを時系列に表現することができる．また，オブジェクト図ではフィールドを介したオブジェクト間の参照をグラフとして表現することができる．これらの図は，プログラムの設計時に作成されたり，プログラムの実行時の情報から手動もしくは自動で構成されて，保守・再利用作業におけるプログラム理解を支援するために用いられる．しかしながら，これらの既存の表現手法で複雑なプログラムの動作のすべての側面を可視化できるわけではない．

そこで本研究では，プログラムの動的側面としてメッセージを介したオブジェクトの移動に関する協調動作に着目し，その協調動作の複雑性に対応するため，動的グラフを用いた可視化手法の構築を目指す．本稿では，まずオブジェクトの移動に関する協調動作の複雑性を交替複雑度として定式化する．さらに，交替複雑度がその協調動作の複雑性をどの程度表しているかを評価する被験者実験を行い，被験者が評価したプログラムの複雑度と交替複雑度が強い相関を示すことを確認した．また，オブジェクトの移動に関する協調動作の全体像把握の容易化を目的に，動的グラフに基づく可視化手法を考案し，そのプロトタイプをデルタ抽出を用いて開発した．今後，このプロトタイプを実際のソフトウェアにおける協調動作の把握に役立つよう完成させていく予定である．

2. 研究の動機

オブジェクト指向プログラムの動作には，さまざまな側面がある．オブジェクトは実行時に生成され，さまざまなメッセージを送受信しながら，その状態を変化させる．また，オブジェクトはフィールドを介して他のオブジェクトから参照され，オブジェクト全体として巨大なグラフを構成する．このオブジェクトグラフの形状はプログラムの実行に伴って変化し，プログラムの制御はメッセージの送受信を通じて，さまざまなオブジェクトの間を移動する．このようなプログラムの内部動作を，さまざまな側面から可視化する表現手法があるが，既存の表現手法で複雑なプログラムの動作のすべての側面を可視化できるわけではない．

本研究が対象とするプログラムの動的側面を説明するための例題プログラムを**図 1** に示す．このプログラムは，`A`，`B`，`C`，`D`，`E` の 5 つのクラスから構成された Java プログラムである．例題プログラム中で `A` クラスの `m()` メソッド（以下，`A#m()` のように書く）が呼び出されたとする．こ

[1] 甲南大学大学院 自然科学研究科
Graduate School of Natural Science, Konan University
[a)] m2024005@s.konan-u.ac.jp
[b)] n-nitta@konan-u.ac.jp

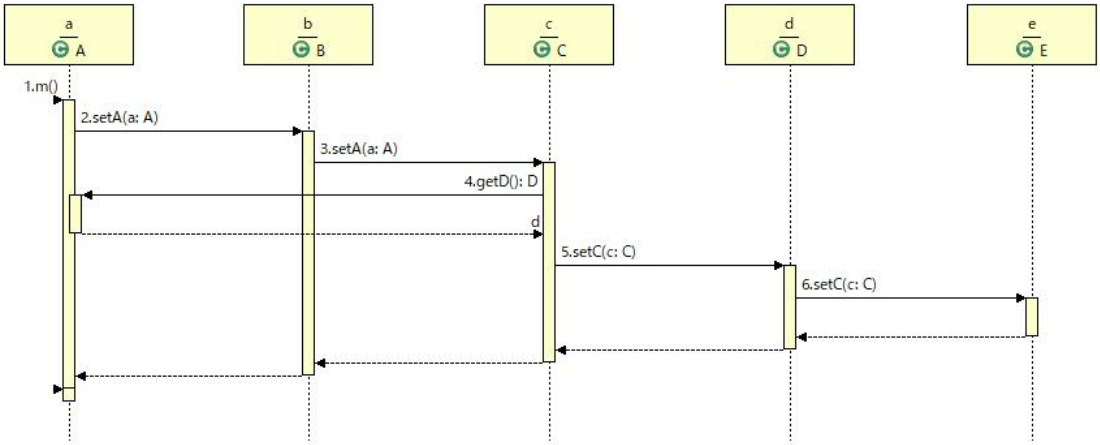

図 3　例題プログラムのシーケンス図

```
1  public class A {
2          B b = new B();
3          D d = new D();
4          void m() {
5                  b.setA(this);
6          }
7          D getD() {
8                  return d;
9          }
10 }
11 public class B {
12         C c = new C();
13         void setA(A a) {
14                 c.setA(a);
15         }
16 }
17 public class C {
18         void setA(A a) {
19                 a.getD().setC(this);
20         }
21 }
22 public class D {
23         E e = new E();
24         void setC(C c) {
25                 e.setC(c);
26         }
27 }
28 public class E {
29         C c = null;
30         void setC(C c) {
31                 this.c = c;
32         }
33 }
```

図 1　例題プログラム

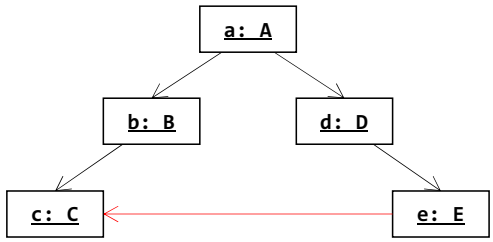

図 2　例題プログラムのオブジェクト図

のとき，メモリ中のオブジェクトは図 2 に示したオブジェクト図の赤線部分がない状態になっている．その後，実行が 31 行目に到達し，その代入文を実行することによって，赤線部分のオブジェクト間参照が生成され，図 2 のような状態になる．ここで C クラスのインスタンスは，B クラスのインスタンスがフィールド c で保持している状態から E クラスのインスタンスの setC() メソッドの引数に渡されるまで「移動」する．このとき，このプログラムは規模が小さいにもかかわらず，この C クラスのインスタンスの移動の経緯を把握することが容易ではないことがわかる．しかしながら，図 2 のようなオブジェクト図では C クラスのインスタンスの移動の経緯を表すことができない．また，図 3 のようなシーケンス図ではオブジェクトの生存線を縦方向の直線で表すため，引数や戻り値によるオブジェクトの受け渡しを明に表現することができない．

3. オブジェクトの移動に関する協調動作

前節で説明したプログラムの実行では，メソッド A#m() の呼び出し直前に B クラスのインスタンスによって保持されていた C クラスのインスタンスが，A#m() の呼び出しによって，E クラスのインスタンスに保持されるところまで移動した．このとき，引数や戻り値を介した受け渡しによって，移動の対象となる C クラスのインスタンスだけでなく，A クラスや D クラスのインスタンスも移動している．また，これらのオブジェクトの移動には，A，B，C，D，E クラスのすべてのインスタンスが関与している．このようにオブジェクトの移動は，移動対象となるオブジェクトを含めた多数のオブジェクトの協調動作によって成り立っている．これを本稿では，**オブジェクトの移動に関する協調動作**という．ここで，たとえ移動対象となるオブジェクト，移動の出発点，移動の終着点がすべて同じであったとしても，それを実現するための協調動作は何通りも考えられることに注意が必要である．例えば図 4 に示すプログラムでは，前節で紹介したプログラム同様，A#m() の呼び出しによって C クラスのインスタンスが，B クラスのインスタンスによって保持されている場所から E クラスのインスタンスに保持される場所まで移動するが，その協調動作の内容

```
1  public class A {
2          B b;
3          D d;
4          void m() {
5                  d.setC(b.getC());
6          }
7  }
8  public class B {
9          C c;
10         C getC() {
11                 return c;
12         }
13 }
14 public class C {
15 }
16 public class D {
17         E e;
18         void setC(C c) {
19                 e.setC(c);
20         }
21 }
22 public class E {
23         C c = null;
24         void setC(C c) {
25                 this.c = c;
26         }
27 }
```

図 4 オブジェクトの移動に関するもう 1 つの協調動作

は前節で紹介したプログラムとは異なる．

4. 協調動作の複雑性指標

図 1 と図 4 のプログラムは，移動対象となるオブジェクト，移動の出発点，移動の終着点はすべて同じであるが，それを実現するための協調動作が大きく異なっており，明らかに図 1 のプログラムにおける協調動作の方が複雑に見える．

そこで本節では，この複雑性が何によってもたらされるものなのかについて考察していく．まず最初に，図 4 のプログラムがなぜ複雑ではないかについて考える．これら 2 つのプログラムにおいて，ともに C クラスのインスタンスが移動対象であったことを思い出して欲しい．図 4 のプログラムでは，プログラム全体を通して引数や戻り値によって受け渡しされるオブジェクトは，この C クラスのインスタンスのみであり，逆に C クラスのインスタンスがメッセージの送信者や受信者になることはない．すなわち協調動作に関わるオブジェクトが，メッセージを通じて送受信される役割と，メッセージを送受信する役割に分けられており，どのオブジェクトもプログラム中でその役割から外れることはない．一方，図 1 のプログラムでは C クラスのインスタンスに加え，A クラスや D クラスのインスタンスも引数や戻り値に出現し，逆に C クラスのインスタンスがメッセージの受信者 (14 行目) や送信者 (19 行目) にもなっている．以上のことから，本稿では協調動作に関わるオブジェクトのうち，移動対象となるオブジェクトがメッセージの送受信者として出現する回数と，移動対象でないオブジェクトがメッセージの引数や戻り値として出現する回数の合計を**交替複雑度**とし，これをオブジェクトの移動に関する協調動作の複雑性を表す指標として定義する．**表 1** に図 1 および図 4 のプログラムにおける，移動対象となるオブジェクトおよび移動対象でないオブジェクトの各役割での出現回数をまとめたものを示す．表より，図 1 と図 4 のプログラムの交替複雑度が，それぞれ 4 と 0 であることがわかる．

5. 複雑性指標の評価実験

前節で説明した協調動作の複雑性指標と，プログラムを理解する上での困難さがどの程度一致しているかを確認するために，被験者実験を行う．本実験は，リサーチクエスチョンを満たすように設計した実験方法に基づいて，被験者によるプログラム理解に関した 2 つの実験を行う．

5.1 リサーチクエスチョン

まず，オブジェクトの移動に関する協調動作の複雑性指標が，プログラムを理解するときの難しさの度合いを表すという仮説を検証する．この仮説を検証するため，交替複雑度と，被験者が実際のプログラムを理解する上で感じる困難さに関係性があるか，また，関係性がある場合にどの程度一致しているのかについて調査を行う．プログラム理解に影響を与える要因が何かを調べるには，ソースコードの行数（LOC）やクラス名の分かり易さなど，さまざまな条件を変えた膨大な数の例題プログラムを用意し，それらすべてを被験者に評価してもらわなければならない．しかしながら，そのような実験は被験者の多大な負担となり，現実的ではない．そこで，プログラム理解に影響する可能性があるプログラムの属性を限定し，交替複雑度を変えた複数種類の小さなプログラムを作成して被験者実験を行う．我々が作成したプログラムを仮想課題と呼び，仮想課題における複雑性指標の有効性を確認するためのリサーチクエスチョン（RQ）を以下に示す．

RQ1 オブジェクトの移動に関する複数の協調動作を比較したときに，それらの捉えにくさの傾向は人によって変わらないか．

RQ2 仮想課題において，交替複雑度はオブジェクトの移動に関する協調動作の捉えにくさを表しているか．

RQ1，RQ2 で用いる仮想課題は，意図的に交替複雑度が高くなるよう構成することができる．しかしながら実際のプログラムにも，そのような高い交替複雑度を持つものが存在するかどうかは定かではない．そこで，実際のプログラムの交替複雑度を調査するためのリサーチクエスチョンを以下に考える．

RQ3 仮想課題でみられる高い交替複雑度を持つような協調動作は実際のプログラムでもみられるか．

これまでに立てたリサーチクエスチョンでは，プログラムの規模が小さく，クラス名が意味をもたない仮想課題を対

表 1　図 1 および図 4 のプログラムにおける移動対象オブジェクトと移動対象外オブジェクトの出現回数

		移動対象オブジェクトの出現回数	移動対象外オブジェクトの出現回数
図 1 のプログラム	引数または戻り値	2	3
	送信者または受信者	1	4
図 4 のプログラム	引数または戻り値	3	0
	送信者または受信者	0	3

象とした被験者実験しか行えないため，複雑性指標が一般のプログラムに対しても有効であるかどうかはわからない．そこで，実際のプログラムから抜き出したソースコードを用いて，実用規模のプログラムにおける複雑性指標の有効性を確認する被験者実験を行う．実際のプログラムから抜き出したソースコードを実課題と呼び，RQ1〜RQ3 の結果および実課題を対象とした被験者実験を総合的に考慮して，以下のリサーチクエスチョンを考える．

RQ4　実際のプログラムにおいても交替複雑度はオブジェクトの移動に関する協調動作の捉えにくさを表しているか．

これらのリサーチクエスチョンを踏まえて，仮想課題と実課題それぞれを対象にした実験方法について 5.2.1 節と 5.3.1 節で説明する．

5.2　仮想のプログラムを対象とした実験

5.2.1　実験方法

本実験では，被験者が仮想課題を理解するときの困難さをアンケート評価してもらう．被験者はクラウドソーシングサービスで募集し，Java の実務歴 3 年以上の 12 名を対象に実験を行った．実験に関するすべての作業を Web 上に提示し，2020 年 9 月 21 日〜10 月 14 日の期間中，被験者が自由に作業を進められるようにした．仮想課題として図 1，図 4 に示したプログラムを含む 6 種類の小規模なプログラムを作成した．仮想課題の交替複雑度は可能な限り均等に分布するよう，0〜4 の交替複雑度を持たせた．プログラム理解に影響するプログラムの属性が最小となるように仮想課題中の以下に示す要素を共通にした．

- クラスの数と名称
- 協調動作に入る直前のオブジェクト図
- 協調動作に関するオブジェクトの数
- 移動対象となるオブジェクト，移動の出発点，終着点

仮想課題のソースコードは，Web ブラウザ上の 1 ページにすべてのクラスを列挙する形で被験者に提示した．6 種類ある仮想課題を読解する順番は順序効果を考慮し，最初に読解するプログラムは共通，その後読解する順番を 2 通り作成し，Java の実務歴の総計が等しくなるよう被験者を 2 つのグループに割り当てた．アンケートは Google フォームで作成し，各仮想課題を読解した後に回答する仮想課題個別アンケートと，すべての仮想課題を読解した後に回答する仮想課題共通アンケートの 2 種類を用意した．仮想課題個別アンケートでは，6 種類の仮想課題に対して，以下の質問を行う．

- ソースコードを読んで，「別々の場所にあった C クラスのインスタンスと E クラスのインスタンスが，どのようにして関連付けられたか?」を把握するのは困難でしたか?

仮想課題共通アンケートでは，すべての仮想課題を読解した後に，困難であった理由を提示された選択肢から複数個選択してもらう．選択肢の内容は 5.2.2 節の結果にて示す．

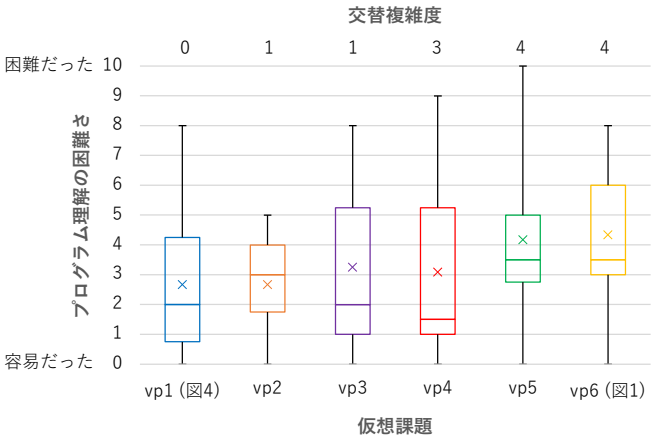

図 5　仮想課題個別アンケートの集計結果

5.2.2　結果

被験者が仮想課題を理解するときに感じた困難さの度合いについて，仮想課題個別アンケートの集計結果を**図 5** に示す．図 5 は仮想課題を交替複雑度の昇順に並べた箱ひげ図であり，縦軸は被験者が仮想課題を理解するときに感じた困難さを 11 段階で表している．以下，仮想課題の交替複雑度が低い順に vp1〜vp6 とする．ここで vp1 と vp6 の仮想課題は，それぞれ図 4 と図 1 に示したプログラムである．また vp4 は，2 つのグループ共通で最初に読解するプログラムである．図 5 に示したプログラム理解の困難さの度合いが，仮想課題間で有意差があるかどうかを t 検定と u 検定（Wilcoxon の符号付順位和検定）の両側検定により確認した．それぞれの検定結果における p 値を**表 2** に示す．表 2 中では，有意水準が 5%以下の場合に有意差ありとみなし，太字で示している．

被験者が仮想課題を理解するのが困難と感じた理由について，仮想課題共通アンケートの集計結果を**図 6** に示す．図 6 に示した選択肢以外では「インスタンスの依存関係をイメージするとシンプルではなく複雑に感じた」や「仮想

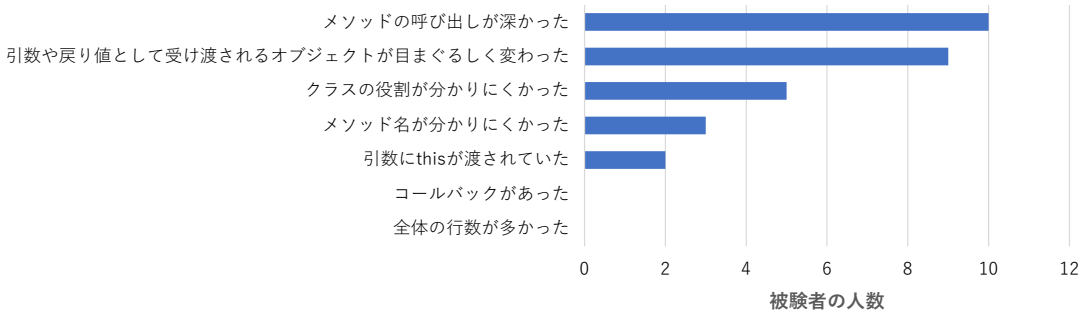

図 6 仮想課題共通アンケートの集計結果

表 2 仮想課題個別アンケートの検定結果

		vp1	vp2	vp3	vp4	vp5	vp6
vp1	t 検定		1	0.09	0.24	**0.007**	**0.001**
	u 検定		0.93	0.93	0.25	**0.015**	**0.005**
vp2	t 検定			0.28	0.55	**0.01**	**0.0007**
	u 検定			0.29	0.48	**0.02**	**0.008**
vp3	t 検定				0.73	0.14	**0.04**
	u 検定				0.79	0.15	**0.03**
vp4	t 検定					**0.04**	**0.03**
	u 検定					0.05	**0.03**
vp5	t 検定						0.71
	u 検定						0.68

表 3 仮想課題におけるプログラムの属性と理解の困難さの相関

属性	vp1	vp2	vp3	vp4	vp5	vp6	相関
呼び出しの最大深さ	3	2	3	4	4	5	0.83
交替複雑度	0	1	1	3	4	4	0.87
this の出現回数	0	0	0	1	2	2	0.92
LOC	18	18	18	20	22	22	0.92
困難さの平均	2.67	2.67	3.25	3.08	4.17	4.33	

課題のロジックをそのようにする理由が分からなかった」といった意見をいただいた．

仮想課題個別アンケートと仮想課題共通アンケートの結果から，プログラム理解に影響を与えうるプログラムの属性と，被験者が仮想課題を理解するときに感じた困難さ間の統計学的な相関を調べる．それぞれの仮想課題における属性の値と，プログラム理解における困難さの平均値とのピアソン相関係数を求め，その結果を**表 3** に示す．表 3 においてプログラム理解を困難にしている属性は，図 6 に示した仮想課題共通アンケートの選択肢の上から 1，2，5，7 番目とそれぞれ対応している．

5.3 OSS を対象とした実験

5.3.1 実験方法

本実験では，まず実際のプログラムの交替複雑度を調査する．そして，そのプログラムの機能における理解を実課題として被験者に提示し，理解するときの困難さをアンケート評価してもらう．交替複雑度の調査および被験者実験に用いる実課題として Java で書かれた 2 種類のオープンソースソフトウェア（OSS）から，それぞれ 2 機能ずつを対象とする．OSS としては，UML モデリングツールである ArgoUML ver. 0.34[*1] と，2 次元図形エディタである JHotDraw ver. 7.6[*2] を選び，ArgoUML からはクラス選択機能とクラス削除機能を，JHotDraw からは図形選択機能と図形移動機能を選ぶ．

被験者実験の実験方法について説明する．被験者と実施期間は，5.2.1 節に示した実験と同様である．実課題のソースコードとして，それぞれの機能に関する部分のみを抜き出し，そのソースコードを Eclipse のデバッガに似せた Web ブラウザ上のデバッガ画面で課題毎に提示する．被験者が呼び出しスタックの内容について確認したり，実行の流れをたどったりする操作ができるようにした．4 種類ある実課題を読解する順番は順序効果を考慮し，アプリケーション上の機能の操作順とは逆順にした上で，ArgoUML と JHotDraw の順番を入れ替えた 2 通り作成し，5.2.1 節に示した実験と同様 2 つのグループに割り当てた．アンケートは Google フォームで作成した．各実課題を読解した後に回答するアンケートを実課題個別アンケートと呼び，アンケートの質問内容の例を以下に示す．

- ソースコードを読んで，「配置された図形が ArgoUML のシステム内部でどのように管理され，選択機能の実行によってどのように取り出されて，どのように選択図形として登録されるか?」を把握するのは困難でしたか?（ArgoUML クラス選択機能）

5.3.2 結果

実際のプログラムにおける交替複雑度の調査結果と，被験者が実課題を理解するときに感じた困難さの度合いについての実課題個別アンケートの集計結果を**図 7** に示す．図 7 は実課題を交替複雑度の昇順に並べた箱ひげ図であり，縦軸は被験者が実課題を理解するときに感じた困難さを 11 段階で表している．以下，実課題に用いた ArgoUML

[*1] https://github.com/argouml-tigris-org/argouml
[*2] https://sourceforge.net/projects/jhotdraw/

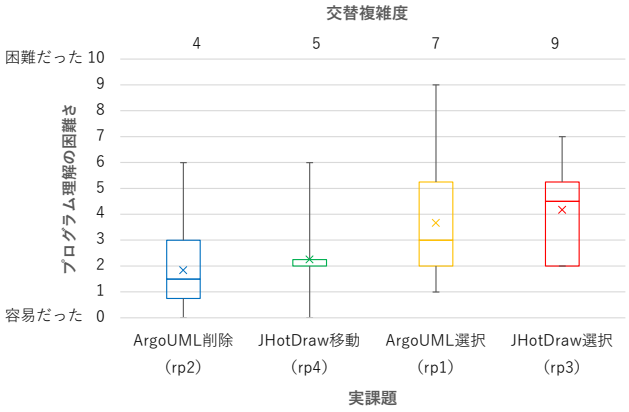

図 7 実課題の交替複雑度と個別アンケートの集計結果

表 4 実課題個別アンケートの検定結果

		rp2	rp4	rp1	rp3
rp2	t 検定		0.24	**0.003**	**0.008**
	u 検定		0.26	**0.008**	**0.009**
rp4	t 検定			**0.04**	**0.02**
	u 検定			**0.03**	**0.03**
rp1	t 検定				0.45
	u 検定				0.59

表 5 実課題におけるプログラムの属性と理解の困難さの相関

属性	rp1	rp2	rp3	rp4	相関
呼び出しの最大深さ	3	11	3	5	0.62
交替複雑度	4	7	5	9	0.98
this の出現回数	0	1	0	0	0.41
LOC	33	102	34	97	0.96
困難さの平均	1.83	3.67	2.25	4.17	

のクラス選択機能，クラス削除機能，JHotDraw の図形選択機能，図形移動機能の順に rp1〜rp4 とする．図 7 に示したプログラム理解の困難さの度合いが，実課題間で比較して有意差があるかどうかを t 検定と u 検定の両側検定により確認した．それぞれの検定結果における p 値を**表 4** に示す．表 4 中では有意水準が 5%以下の場合に有意差ありとみなし，太字で示している．

実課題個別アンケートの結果から，プログラム理解に影響を与えうるプログラムの属性と，被験者が実課題を理解する上で感じる困難さの統計学的な相関を調べる．実課題におけるそれぞれの属性の値と困難さの平均値とのピアソン相関係数を求め，その結果を**表 5** に示す．

6. 考察

リサーチクエスチョン毎に考察を行う．まず RQ1 についてであるが，図 5 より，同じ仮想課題でも被験者によって感じる困難さにばらつきがあることがわかる．各被験者の回答を詳細に見ていくと，約半数の被験者の仮想課題に対する捉えにくさの傾向はよく一致していたが，残りの被験者の傾向はまちまちであった．したがって RQ1 については，全体として一定の傾向は認められるが，必ずしもすべての被験者の傾向が一致しているわけではないといえる．次に RQ2 であるが，仮想課題について，図 5 より，交替複雑度の増加に伴って被験者が感じるプログラム理解の困難さも増加する傾向にあることがわかる．また表 2 より，交替複雑度の差が大きくなると，被験者が感じるプログラム理解の困難さに有意差が現れることがわかった．プログラム理解の困難さに影響を及ぼす可能性があるプログラムの属性は，メソッド呼び出しの最大深さなど交替複雑度以外にも考えることができる．図 6 に示したアンケート結果より，これらの属性の中で，メソッド呼び出しの最大深さと，交替複雑度が，仮想課題の理解を困難にしている主たる要因であると考えられる．さらに，表 3 で，交替複雑度の方がメソッド呼び出しの最大深さよりも相関係数が高かったことから，RQ2 は成り立っているといえる．RQ3 については図 7 より，4 つの実課題のうち 3 つが仮想課題よりも高い交替複雑度を持っていることから，成り立っているといえる．RQ4 に関しては，実課題実験に用いた課題数が多くないため一般的にはいえないが，図 7 および表 4 において，交替複雑度とプログラム理解の困難さの間に明確な関係があることを確認できること，表 5 において，交替複雑度とプログラム理解の困難さとの間に非常に高い相関が見られることから，成り立っていると考えられる．以上の結果より，実際のプログラムの機能理解において，オブジェクトの移動に関する協調動作を捉えにくい場合が少なくなく，交替複雑度はその捉えにくさを表す良い指標の 1 つであるといえる．

7. 動的グラフに基づく可視化手法

4 節ではメッセージを通じて送受信される役割のオブジェクトと，メッセージを送受信する役割のオブジェクトが，プログラム中で本来の役割から外れる度合いを交替複雑度として定義した．本節では，交替複雑度の高いプログラムをどのように分かり易く可視化するかについて考える．高い交替複雑度を持つ場合，移動対象のオブジェクトだけでなく多くのオブジェクトがメッセージを通じて送受信されることになる．しかしながら 3 節で述べたように，オブジェクト間のメッセージのやり取りを時系列に表すシーケンス図ではオブジェクトの生存線が垂直に伸びているため，メッセージを介したオブジェクトの受け渡しを表すのに向いていない．そこで，オブジェクトの移動に関する協調動作を動的グラフに基づき可視化する表現手法として**マグネットモデル**を提案する．以下にマグネットモデルの定義を示す．

(1) プログラムの実行時の状態変化をアニメーションで表す．
(2) 協調動作に関わるオブジェクトを楕円で表し，内部にクラス名を下線付きで記す．オブジェクトは，協調動作に入る直前の参照構造にしたがって特定の形状に配

置し，それを元の位置とする．
(3) 協調動作に関わる各メソッド実行 m を矩形で表し，内部にメソッド名を記す．m を表す矩形は，m が実行されているオブジェクトの内部に配置する．メソッド実行が終了し，呼び出しスタックから取り除かれた場合，m を表す矩形は削除する．
(4) メソッド実行間の呼び出し関係を破線矢印で表す．
(5) フィールドによるオブジェクト間の参照関係を楕円間の実線矢印で表し，実線矢印横にフィールド名を記す．
(6) スタック中のメソッド実行 m において，引数，局所変数，無名変数などによって局所的に参照されているオブジェクトを o とおく．o の位置を m を表す矩形に接するように移動させ，かつ，m が実行されているオブジェクト o' の内部に可能な限り含まれるように配置する．必要に応じて o' のサイズを大きくする．
(7) スタック中の呼び出し先のメソッドが実行されているオブジェクトを可能な限り呼び出し元のメソッドが実行されているオブジェクトの手前に描画する．
(8) どのメソッド実行からも参照されなくなったオブジェクトは元の位置に戻す．
(9) 複数のメソッド実行によって参照されているオブジェクトは，最も呼び出し先にあるメソッド実行に接するように配置する．
(10) 上記条件を同時に満たすことができない場合，最も呼び出し先にあるメソッド実行に関する条件を優先する．

マグネットモデルはメソッドの呼び出し，メッセージを介した引数や戻り値によるオブジェクトの受け渡しを全てアニメーションで表現するため，高い交替複雑度を持つ協調動作を簡潔に表現することができる．マグネットモデルはメソッド実行を電磁石，オブジェクトを金属球，オブジェクト間参照をワイヤと見立てたときに，最も呼び出し先にあるアクティブなメソッド実行に対応する電磁石に，局所的に参照されているオブジェクトに対応する金属球が引き付けられる様子の類推として定義した．マグネットモデルはプログラムの複数実行時点を同時に提示できないという短所がある．

8. プロトタイプシステムの実装

前節で提案したマグネットモデルを用いて任意の Java プログラムの実行時の動きを可視化するツール MagnetRON のプロトタイプを開発した．本節では MagnetRON（以下，本ツールと略）の実装の詳細について説明する．本ツールはオブジェクトの移動に関する協調動作をアニメーションとして可視化するために，プログラムの実行時の情報を必要とする．さらに，膨大な実行時の情報から協調動作に関する情報を抜き出す必要がある．そこで本研究室で提案している**デルタ抽出** [6] と呼ばれる動的解析技術 [9] を用いて，プログラムの実行時の情報からオブジェクトの移動に関する情報を抜き出す．デルタとは 2 つのオブジェクトが関連付けられた経緯を表す動的情報である．例えば，図 1 および図 4 のプログラムの `C` クラスのインスタンスと `E` クラスのインスタンスに対してデルタ抽出を行うと，それらが関連付けられた経緯のソースコードを行単位で抜き出すことができる．またデルタ抽出は，図 2 に示したようなオブジェクトグラフも同時に抜き出す．ここで図 2 より，デルタ抽出によって抜き出されるオブジェクトの移動は，`C` クラスのインスタンスが移動対象となるオブジェクト，`B` クラスのインスタンスによって保持されていた場所が移動の出発点，`E` クラスのインスタンスに保持される場所が移動の終着点に相当することがわかる．このように，デルタ抽出によってオブジェクトの移動に関する協調動作を抜き出せるため，デルタ抽出ツールと連携して本ツールを構築する．そして，デルタ抽出によって抜き出された情報を基にアニメーションの自動生成を行う．デルタ抽出ツールと連携するにあたり，ソースコード行より細かい単位のエイリアスを出力できるようデルタ抽出ツールの改変を行った．なお，7 節で示したマグネットモデルの定義の（2）では，オブジェクトを配置する特定の形状については明記していない．本ツールはデルタ抽出で抜き出したオブジェクトを図 2 のような三角形の形状に配置する．

本ツールには，移動対象となるオブジェクトおよび移動の終着点，対象プログラムの実行時情報を時系列に記録したトレースファイルを入力し，出力としてグラフをアニメーションで表示する．なお，デルタ抽出を用いた場合，移動の出発点は移動の終着点から一意に定まる．任意の出発点からのアニメーションを得たい場合は，複数回 MagnetRON を適用しなければならない．出力に必要なグラフの描画には，グラフ描画ライブラリ JGraphX[*3] を用いた．本ツールの動作画面を**図 8** に示す．

9. 関連研究

9.1 ソフトウェアメトリクスに関する関連研究

交替複雑度は複雑度を表すソフトウェアメトリクス [3] の 1 つである．複雑度を表す代表的なメトリクスとして，サイクロマティック複雑度 [5] が挙げられるが，サイクロマティック複雑度は主に制御構造の複雑性を評価するものであり，図 1 のプログラムのような協調動作の複雑性を評価するのには適していない．オブジェクトの移動に関する協調動作は，プログラム実行時の振る舞いであるため，その複雑性を評価するには，動的なソフトウェアメトリクス [1] が有効であると考えられる．複雑性に関する動的なソフトウェアメトリクスとして，Munson ら [2] は，コンポーネントの複雑性をコンポーネントが実行される頻度で重み付けして平均を取り，システム全体の複雑性として評価するメ

[*3] https://github.com/jgraph/jgraphx

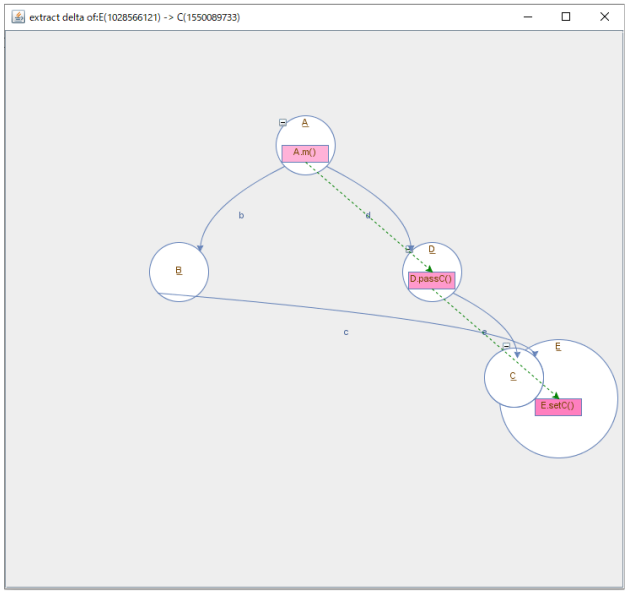

図 8　可視化ツールの動作画面

トリクスを提案した．また，Yacoub ら [8] は，実行シナリオにしたがってステートチャートを遷移させた場合のサイクロマティック数を実行シナリオが実行される頻度で重み付けして平均を取ったものを，オブジェクトの複雑性を評価するメトリクスとして提案した．いずれのメトリクスにおいても，評価対象が協調動作ではないこと，および引数や戻り値によるオブジェクトの受け渡しを考慮していないことから，交替複雑度とは異なる．

9.2　プログラムの可視化に関する関連研究

UML においては，主にオブジェクト図とシーケンス図を用いて，プログラムの動的側面が可視化される．しかしながら 2 節で説明したように，オブジェクトの移動に関する協調動作を可視化するには，オブジェクト図やシーケンス図だけでは不十分である．動的オブジェクトプロセスグラフ [7] は，制御フローグラフから指定したオブジェクトに関する部分のみを抜き出したものであり，特定のオブジェクトの移動に関する情報を表現するために用いることができる．マグネットモデルも特定のオブジェクトの移動に関する情報を表現したものであるが，指定したオブジェクトだけでなく協調動作に関わるすべてのオブジェクトに関する情報もまとめて表現することができる．オブジェクト指向プログラムの動的な可視化手法として，山崎ら [10] は，オブジェクト図のアニメーションによる表現手法を提案している．この手法では，オブジェクト指向プログラムの詳細な実行時情報を表現することができるが，引数や戻り値で受け渡しされるオブジェクトがオブジェクト ID で表現されるため，メッセージの送受信者となるオブジェクトと，引数や戻り値で受け渡しされるオブジェクトとの対応が取りにくく，オブジェクトの移動に関する協調動作を表現するには適さない．

10.　おわりに

オブジェクトの移動に関する協調動作の複雑性を評価する指標として交替複雑度を導入し，被験者実験を行って交替複雑度の有効性を確認した．また，オブジェクトの移動に関する協調動作を動的グラフを用いて可視化する手法として，マグネットモデルを考案し，デルタ抽出を利用してプロトタイプシステムの実装を行った．今後，実際のソフトウェアにおける協調動作の理解を支援できるよう，可視化ツールを拡張していきたい．具体的には，デルタ抽出を用いることによるオブジェクトの移動範囲の制限をなくすことを考えている．また，被験者実験を行ってマグネットモデルの有効性を評価することも予定している．

謝辞　実験に協力いただいくとともに有益なご助言をいただいた被験者の皆様に感謝の意を表する．この研究の一部は, 私立大学等経常費補助金 特別補助「大学間連携等による共同研究」によるものである．

参考文献

[1] Jitender K. Chhabra and Varun Gupta, “A survey of dynamic software metrics,” *Journal of Computer Science and Technology*, Vol. 25, No. 5, pp. 1016–1029, (2010).
[2] Taghi M. Khoshgoftaar, John C. Munson and David L. Lanning, “Dynamic system complexity,” in *Proc. Software Metrics Symposium*, pp. 129–140, (1993).
[3] Michele Lanza and Radu Marinescu, *Object-oriented metrics in practice*, Springer, (2006).
[4] Adrian Lienhard, Tudor Gîrba and Oscar Nierstrasz, “Practical object-oriented back-in-time debugging,” in *Proc. the 22nd European Conf. on Object-Oriented Programming (ECOOP’ 08)*, pp. 592–615, (2008).
[5] Thomas J. McCabe, “A complexity measure,” *IEEE Transactions on Software Engeneering*, Vol. 2, No. 4, pp. 308–320, (1976).
[6] Naoya Nitta and Tomohiro Matsuoka, “Delta extraction: An abstraction technique to comprehend why two objects could be related,” in *Proc. 31st IEEE International Conference on Software Maintenance and Evolution (ICSME’ 15)*, pp. 61–70, (2015).
[7] Jochen Quante and Rainer Koschke, “Dynamic object process graphs,” *Journal of Systems and Software*, Vol. 81, Issue. 4, pp. 481–501, (2008).
[8] Sherif Yacoub, Hany H. Ammar and Tom Robinson, “Dynamic metrics for object oriented designs,” in *Proc. Software Metrics Symposium*, pp. 50–61, (1999).
[9] 石尾隆, “プログラムの動的解析,” コンピュータソフトウェア, Vol. 29, No. 1, pp. 47–60, (2012).
[10] 山崎翔, 久保田吉彦, 紫合治, “オブジェクト図のアニメーション,” ソフトウェアエンジニアリングシンポジウム, pp. 129–136, (2015).