初学者の協調プログラミングにおける インタラクションの分析

加藤 優哉[1,a)]　松澤 芳昭[2,b)]　酒井 三四郎[1,c)]

概要：初学者の協調プログラミングにおけるインタラクションの分析を行った．本稿における協調プログラミングとは，(1) **collective contribution**：能力が一律ではないグループのメンバ個々人が実力に見合った貢献ができていること，(2) **productive interaction**：完全な分業ではなく，他のグループメンバが書いたソースコードを読み，それを拡張させるようなプログラミングが行われていること，である．協調プログラミングを支援するために「独立同期モデル」に基づくシステム（CheCoPro）を実装した．CheCoPro をプログラミング入門講義において導入し，文科系の学生 95 名に対して実験を行った．システムのログから，グループメンバー間のプログラムや素材ファイルのやり取りを表すインタラクション図を作成し分析を行った．結果，協調プログラミングの定義を共に満たすようなインタラクションパターンが見られた．アンケートの結果，CheCoPro 利用群と非利用群を比較して，遠慮の減少と貢献の増加にそれぞれ有意な差が見られた．

キーワード：協調学習，プログラミング教育，初学者，インタラクション

1. はじめに

プログラミング導入教育において，学生はグループで協力してプログラムを記述する機会を与えられることがある [1][2]．本稿ではこのようなプログラミングを協調プログラミングと呼ぶ（3.2 節にて詳説）．

しかしながら，実際の講義においてプログラミング初学者がグループでのプログラミングを行う上で困難なことがある．我々は講義で 1）プログラムの大半をグループでプログラミングが最も得意な学生（ドライバ）が記述してしまい，その他の学生（フォロワ）はプログラムを書かない，2）グループメンバーが完全にタスクを分割して作業してしまう（例えば，複数の小規模なサブゲームから成るゲームをそれぞれ完全に独立して製作するような）という現象を観察してきた．

ファイルの共有方法にも課題がある．プログラミング導入講義の学生は，例えば git のような構成管理ツールについて学ぶ時間はほとんどない．それにも関わらず，既存の構成管理ツールはプロフェッショナル向けにデザインされているため，学生がツールを使いこなすことは難しい．

以上を背景に，本研究では初学者向けの入門教育における協調プログラミングの支援と分析を試みる．本論文では特に，実際の教育現場における学習者のインタラクションパターンから，初学者の協調プログラミングの実態解明を試みる．

2. 先行研究

2.1 ソフトウェア開発における協調作業のモデル

プログラミング/ソフトウェア開発における協調作業のモデルは，ソフトウェア工学の分野で 30 年以上にわたって議論され続けてきた．中にも，ウォータフォールモデルとアジャイルモデルの 2 つのモデルがある．ウォータフォールモデルは要件定義，デザイン，実装，テストのようなプロセスを完全に分担するモデルであり，30 年以上にわたって最も有力なモデルであった．対照的にここ 10 年では，アジャイルモデルの人気が上昇している．アジャイルモデルは，柔軟で創造的なソフトウェア開発が要求される産業的な開発においても，状況変化や新しいソフトウェア開発の進化に適している．特に，Scrum[3] はソフトウェア開発方法論の中で現在最も人気のある方法である．Scrum は知識創造理論 [4] がベースとなっている．従って，ウォー

[1] 静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University
[2] 青山学院大学社会情報学部
School of Social Informatics, Aoyama Gakuin University
[a)] kato@sakailab.info
[b)] matsuzawa@si.aoyama.ac.jp
[c)] sakai@inf.shizuoka.ac.jp

ターフォールモデルとは正反対の協調作業が行われる．例えば，プロジェクトマネージャによる管理に代わって，自己組織化プロセスが推奨されることが挙げられる．Scrumでは，役割分担をするのではなく，作業を共有し暗黙知を共有することが推奨される．

現在，プロフェッショナルと教育現場の双方で，協調プログラミングでの各モデルの利点が議論されている．我々のモデルは主にアジャイルモデルと知識創造理論に基づいている．近年，参加者間の論理的なインタラクションと役割やリーダーの頻繁な変更が，オープンソースソフトウェア開発コミュニティのような創造的なコミュニティで観察された [5]．教育現場でも同様の現象が観察されている [6]．その他の研究においても，協調プログラミングにおいて学生間のインタラクションの重要性が主張されている．平井らは，プログラミング学習について参加者が意見交換，競合，交渉，合意形成等を繰り返し，グループの合意としての成果を出すことを協調プログラミング学習と定義している [7]．

実践的な観点から，ペアプログラミングとその利点について議論され続けている．ペアプログラミングはアジャイルモデルの前身である eXtreme Programming（XP）[8] の12 のベストプラクティスのうちの 1 つである．Cockburn らは協調プログラミングのモデルとして，ペアプログラミングを拡張した”side by side プログラミング” を提案した [9]．Goldman らは，リアルタイム共同コーディング環境の使い方として，（1）授業でのプログラミング，（2）テスト駆動ペアプログラミング，（3）マイクロアウトソーシング，の 3 つを提案した [10]．しかし，協調プログラミングの部分的な成功にとどまり，結果は，教育での小規模のチームにおいて協調学習の促進に成功したのみであった．

2.2 協調プログラミングの支援ツール

CSCW(Computer Supported Cooperative Work) や CSCL(Computer Supported Collaborative Learning) という分野で，グループでのプログラミングを支援するために多くのツールが提案されてきた．

Collabode は教育目的でデザインされた最新のツールである [10]．Collabode は，複数の学生が同時にコードを編集可能なリアルタイム同期環境である．Vandeventer らも同様なツールを提案している [11]．技術的には，文書のための同時編集環境のライブラリである EtherPad を基にしている．このライブラリは Google Doc にも用いられている．同様なツールとしては Saros がある [12]．Saros は Ecipse のプラグインとして実装されている．このツールは指定したメンバーの視野（スクロールやファイル選択）を追従する機能が実装されている．Salinger らはグループメンバーの意識共有を支援し，コードレビューや遠隔でのペア・パーティプログラミングの際に有用であると主張している．

しかし，リアルタイム同期モデルは営利的なソフトウェア開発やオープンソースプロジェクトのようなプロフェッショナル向けの協調プログラミングではほとんど使われていない．実開発現場では，プロフェッショナルは CVS や subversion，Git のような SCM（Source Configuration Management）支援ツールを 30 年以上使っている．現在では，Git / Github がオープンソースコミュニティでのスタンダードなツールである．これらのツールは最新のツールにも関わらず，単にブランチ＆マージモデルを支援しているだけであり，リアルタイムシェアリングはできない．

現在，教育目的でのリアルタイム同期モデルとブランチ＆マージモデルの利点は明らかになっていない．普遍的な文書の記述において，リアルタイム同期環境の利点の解明に取り組んでいる研究は存在する [13][14]．André らは，規則的な作業においては利点を有するが，複雑な創造的なタスクを同時に取り掛かることには不利であると主張している [15]．一方で，プログラミングの導入教育において，SCM を取り入れて成功したという報告はない．我々は，リアルタイム同期モデルでは学生の編集が直接的にグループの作品に影響を与えてしまうことを問題と考えている．その他の問題として，時間を共有する必要があるため授業時間外の共同作業を促進できない点が挙げられる [14]．対照的に，ブランチ＆マージモデルは，遠慮や時間を共有することがないという点で融通がきく．このモデルの問題は，モデルの概念やツールの操作を学ぶために大きな認知的負荷を要することである．特に，一度コンフリクトが発生するとプロフェッショナルでさえも解決に手間がかかるほどである．

以上のように，コミュニケーションや遠隔でのペアプログラミング支援，プロフェッショナル向けの SCM が提案されている．しかし，初学者が使用可能かつ “協調プログラミング”を支援可能なシステムは開発されていない．

3. 問題分析と協調プログラミングの定義

3.1 問題分析

我々は，プログラミング導入講義の最終課題の場で協調プログラミングを実施している．2013 年度の講義で協調プログラミングに関して調査を行い，70 件の有効な回答が得られた．結果，半数の学生がソースコードの結合や役割分担に関して苦労を感じていることがわかった．プログラムを共有する方法として 60%の学生が USB メモリのような外部メモリを用いていることもわかった．

同講義で，グループ内でのプログラミングスキルとソースコードに関する貢献度合いを測定した．結果として，フォロワはドライバの半分以下のコードしか記述していないことがわかった．原因として，フォロワがドライバに比べてプログラミングスキルが劣るだけでなく，ドライバに遠慮

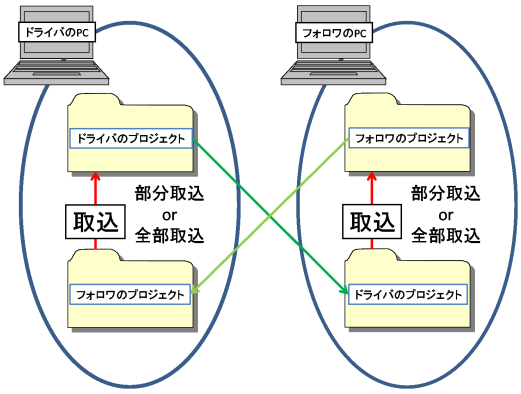

図 1　独立同期モデル

をしてソースコードを記述できなかったことが考えられる．従って，フォロワの意見がプロジェクトに反映されていない恐れがある．

3.2　協調プログラミングの定義

我々の目標は，導入教育環境におけるアジャイルと知識創造理論を基にした協調プログラミングの促進である．特に，実際の講義で発生している 2 つの典型的な問題を解決することに注力すべきであると考えている．1 つは，グループメンバー間のスキルレベルの差である．我々は，グループのメンバーそれぞれに，コード量やプロジェクトに費やす時間について等しくなるように課題に取り掛かることは意図していない．しかし，学生は責任を分けるために，作業量が平等になるように無理に役割分担を行うことがある．2 つ目の問題は，多くのグループについてメンバー同士の仕事が完全に独立していることである．例えば，3 人グループのプロジェクトでメンバーそれぞれが 1 つのミニゲームを作り，それをまとめて 3 つのミニゲームから成るソフトウェアを成果物とするグループがある．この作品は，単に個々の作業の集合に過ぎない．この例の場合，グループメンバー同士でコミュニケーションを取る必要性がほとんどないため，協調プログラミングとは言えない．

以上から本研究における協調プログラミングを以下のように定義する．

collective contribution　能力が一律ではないグループのメンバー個々人が実力に見合った貢献ができていること．

productive interaction　完全な分業ではなく，他のグループメンバが書いたソースコードを読み，それを拡張させるようなプログラミングが行われていること．

4.　提案システム（CheCoPro）

ツールの詳細について既に我々は発表している [16]．本章ではその概要を説明する．

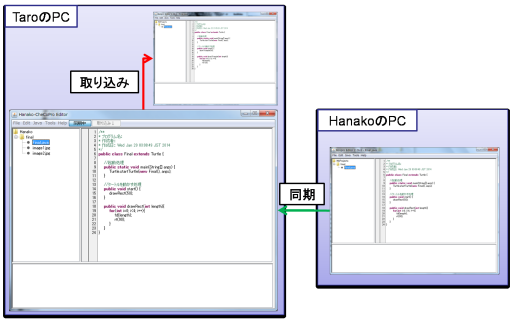

図 2　インタフェース

4.1　独立同期モデル

初学者の協調プログラミングを支援するために既存のモデルを改善した独立同期モデルの概要を述べる．モデルを**図 1** に示す．独立同期モデルは以下の特徴を持つ．

(1) 全てのメンバーが個々の独立したブランチを管理する．
(2) 全てのメンバーが他のメンバーの最新バージョンをリアルタイムに閲覧できる．（他人のコードは直接編集できない．）
(3) 全てのメンバーはいつでも他のメンバーのソースコードを単純操作で取り込むことができる．

4.2　設計目標

独立同期モデルを基に協調プログラミング支援システム CheCoPro（Cheerful Collaborative Programming）を実装した．

CheCoPro は，ファイルをマージする際の複雑性を解消するために，単純な操作でファイルの取り込みが可能なツールである．ファイル共有という観点では，外部ツールを使うことなく行えるようなツールとなっている．

我々は，上記の問題を解決することで，フォロワはグループに貢献することができるようになり，作品もより完成度の高いものになるという仮説を設けている．

4.3　実装

CheCoPro は，プログラミング導入講義のための開発環境上に実装した．インタフェースを**図 2** に示す．基本機能として，同期，観察，取り込み機能の 3 つがある．

同期機能は collective contribution を支援するために実装した機能である．ユーザは他のメンバーのコードビューアを開くことができるが，直接編集することはできない．

観察について，メンバーのソースコードの変更をリアルタイムで観察できる機能を，productive interaction を支援するために実装した．ユーザは自身のエディタ上でソースを編集し，メンバーのコードはビューアとなる他のウィンドウで観察する．

取り込みについて，CheCoPro は 2 つの単純な方法でグループメンバーのソースコードを取り込み可能な機能を実

表 1　課題内容

課題	Java を使った GUI 作品制作
期間	2 週間（2015/01/15〜2015/01/29）
グループ人数	1〜3 名
構成メンバー	受講生同士で任意に決定

表 2　CheCoPro の利用人数

	3 人グループ	2 人グループ
利用群	24 人（8 組）	14 人（7 組）
非利用群	45 人（15 組）	12 人（6 組）

表 3　各群の成績

	平均	標準偏差
利用群	51.2/73	7.4
非利用群	49.3/73	9.0

装している．1 つは全部取込であり，もう 1 つは部分取込である．全部取込は全てのファイルを一括で取り込む取り込み方法である．部分取込はコピー＆ペーストを手動で行う取り込み方法である．

5.　実験方法

プログラミング入門講義の最終課題の場において，CheCoPro を導入して実験を行った．実験の目的は，初学者の協調プログラミングにおけるインタラクションのパターンの抽出と，システムの協調プログラミング支援効果の検証である．

5.1　被験者

文科系の学部 1 年生プログラミング入門講義において CheCoPro を導入した．被験者は，オブジェクト指向の概念を除く Java の基礎的な学習を 4 ヶ月間取り組んでいる．最終課題にエントリした学生から，個人で課題に取り組むことを希望した者を除く 37 グループ（96 名）を被験者とした．被験者は，協調プログラミングに取り組むことは未経験である．

5.2　課題内容

最終課題の要項を**表 1** に示す．最終課題の内容は，Java を使った GUI 作品の制作である．課題に取り組む期間は 2 週間である．グループは 3 名までで，構成メンバーについては任意である．全体に CheCoPro の導入方法と利用方法を 10 分程度で解説し，全員がシステムの導入作業を行った．システム導入は必須としたが，利用は任意とした．導入を必須とした理由は，利用群と非利用群間の最終課題に対するモチベーションのギャップを減らすためである．モチベーションの低い学生は導入を煩わしく感じ，導入段階でシステムの利用を断念することが考えられるためである．最終課題終了後に協調プログラミングに関するアンケートを実施した．

表 4　CheCoPro の取得ログ

	時刻	取込元	取込先	ソースコード
ログイン	✓			
ログアウト	✓			
全部取込	✓	✓	✓	
部分取込	✓	✓	✓	✓

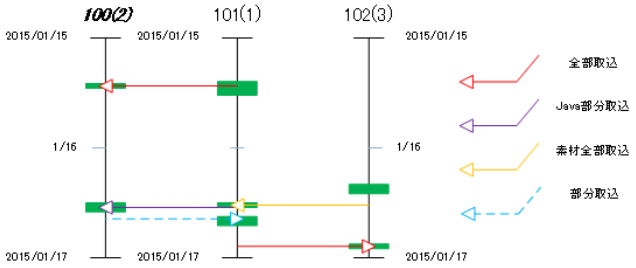

図 3　インタラクション図の例

システムのログとアンケート結果から，CheCoPro を利用して最終課題に取り組んだグループ（利用群）と CheCoPro を利用しなかったグループ（非利用群）に分けて分析を行う．利用群と非利用群の内訳を**表 2** に示す．利用群と非利用群の最終課題に入る前までの成績について**表 3** に示す．成績は課題の提出状況と提出物の内容から，講義担当者と著者を含むティーチングアシスタントによって確定したものである．最終課題までに取得できる成績の最高点は 73 である．表 3 より，利用群と非利用群の成績の差は分析を行うにおいて妥当な小ささであると考えられる．

5.3　インタラクション分析方法

CheCoPro のログとして取得している項目について**表 4** に示す．CheCoPro のログとして，ログイン，ログアウト，全部取込（Java・素材全部取込，Java 全部取込，素材全部取込），部分取込の 5 つの操作について取得している．それぞれタイムスタンプも取得し，取り込み操作に関しては取り込み元と取り込み先のメンバーについても取得している．部分取込に関しては，取り込んだソースコードに関しても取得している．

システムのログを用いてインタラクション図を作成し，分析を行う．インタラクション図とは，グループメンバー同士の取り込みのやりとりを時系列に表した図であり，その例を**図 3** に示す．

縦軸は時間を表す．それぞれの軸の上に記述されている番号はメンバーの ID であり，括弧内の数字はグループ内での成績の順位を表す．ID がイタリックで表示されているメンバーがグループのリーダーである．グループのリーダーは学生間で任意に決めたものである．軸上に帯で表示されている期間がシステムにログインしていた期間である．各矢印は取り込みを表しており，矢印先のメンバーが矢印元のメンバーのソースコード等を取り込んだことを表す．

表 5　グループごとの取り込み回数

グループ ID	全部	Java 全部	素材全部	部分
A	10	0	0	1
B	1	0	2	3
C	7	2	1	2
D	13	0	0	0
E	29	0	3	0
F	2	0	1	3
G	3	0	1	2
H	18	8	7	8
I	9	0	2	3
J	7	8	1	3
K	5	0	2	11
L	18	0	1	4
M	4	0	0	0
N	13	6	7	0
O	9	0	0	1

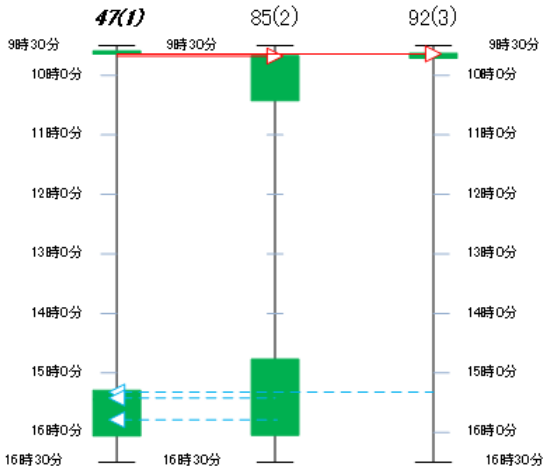

図 4　グループ K のインタラクション図の一部

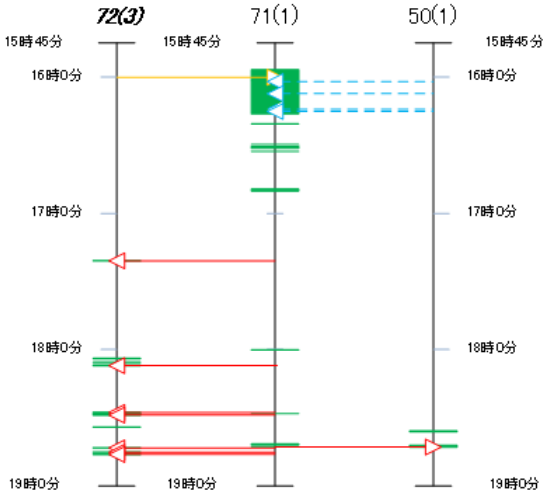

図 5　グループ L のインタラクション図の一部

```
TextTurtle t1 = new TextTurtle("「おはよう！」");
t1.warp(300,600);
if(key() == 13){
  TextTurtle t2 = new TextTurtle("「1時間目は選択授業だよね！」");
  t1.looks(t2);
  if(key() == 13){
    TextTurtle t3 = new TextTurtle(
      "「...あれ？場所はどこだっけ？」1.体育館  2.音楽室  3.美術室");
    t2.looks(t3);
    TextTurtle t31 = new TextTurtle(
      "【数字キーから1か2か3を選んでキーを押してください。】");
    t3.looks(t31);
    if(key() == 49){
      TextTurtle t4 = new TextTurtle("「そっか、体育だよね！がんばろ！」");
    }else if(key() == 50){
      TextTurtle t5 = new TextTurtle("「合唱ね！！」");
    }else if(key() == 51){
      TextTurtle t6 = new TextTurtle("「ううぇーいww今日も漫画書くぞ！」");
    }
  }
}
```

図 6　グループ K の部分取込

```
int scorekeisan(int point){
  int i = 0;
  if(point < 10000){
    i = 4;
  }else if( 10000<= point &&point <20000 ){
    i = 3;
  }else if( 20000<= point && point < 30000){
    i = 2;
  }else if (30000 <=point ){
    i = 1;
  }
  return I;
}
```

図 7　グループ L の部分取込

6.　実験結果

6.1　CheCoPro のログ

ログから得られたグループごとの取り込み回数を**表 5** に示す．全部は Java ファイルと素材ファイルの全部取込を示し，Java 全部と素材全部はそれぞれ Java ファイルのみの全部取込と素材ファイルのみの全部取込を示す．部分は，ソースコードのコピー&ペーストによる部分取込を表す．それぞれの取り込み方法が適宜利用されていることがわかる．

6.2　インタラクション

6.2.1　フォロワの貢献

フォロワが実装したプログラムの一部がドライバに取り込まれ，最終成果物の一部となることがある．その例として，グループ K のインタラクション図の一部を**図 4** に示す．グループ K のフォロワ（85 と 92）がドライバ（47）のプロジェクトを全部取込し，その後，ドライバが各フォロワからプログラムの一部を取り込んでいることが分かる．**図 5** の前半にも同様な操作が見られる．

グループ K の 47 が部分取込によって 92 から取り込んだソースコードを**図 6**，グループ L の 71 が部分取込によって取り込んだソースコードを**図 7** に示す．

グループ K は選んだ選択肢によってシーンが変わるアドベンチャーゲームのようなものを作った．部分取込によって取り込まれたソースは，授業の教室を選ぶという，ゲームの 1 シーンである．47 は，85 から別のシーンに関するソースも部分取込している．以上から，グループ K のフォ

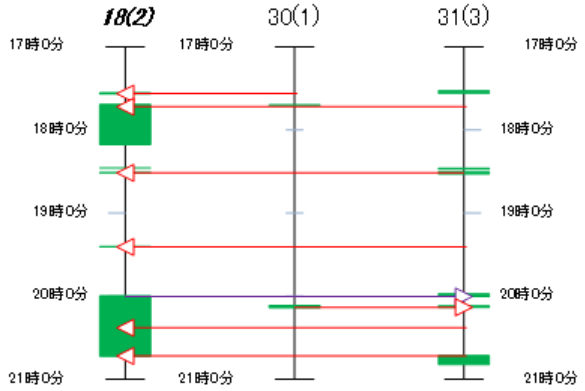

図 8　グループ N のインタラクション図の一部

ロワはそれぞれ担当するシーンを担い，実装をしていたと考えられる．

グループ L はメソッド単位での部分取込が行われている．図 5 から，71 がドライバとなり，50 からはメソッドを受け取り，72 からは素材を受け取るような流れでプロジェクトが進んでいたと想定できる．

6.2.2　締め切り前日の挙動

インタラクションの特徴の 1 つとして，締め切り日の前夜にフォロワがドライバのプロジェクトを繰り返し全部取込するという現象が見られる．グループ L のインタラクション図の一部を図 5 に示す．最終課題締め切りの前日の夕方から夜の部分を抽出したものである．71 はまず 72 から素材を取り込んでいる．その後 50 からソースのパーツを受け取っている．その後 72 が繰り返し全部取込をしている．グループ N にも同様の動きが見られる．グループ N のインタラクション図の一部を**図 8** に示す．18 が何度も 31 から全部取込を行っている．

図 5 と図 8 より，グループ L とグループ N のフォロワはドライバのプログラムを繰り返しテストするテスタとして貢献していると考えられる．ドライバのプログラムを何度も取り込んでいることから，テスト結果をドライバに報告し，ドライバがそれを基に修正をし，再びフォロワが全部取込を行ってテストをするというプロセスを繰り返していると考えられる．

取り込み操作が容易なことから，変更が加わるたびに何度もテストを行う事に関して煩わしさが減少した．テストという観点でフォロワが積極的にプロジェクトに参加することができ，貢献の増加の一因になったと考えられる．

6.2.3　2 人グループの特徴

2 人グループの場合，ドライバ・フォロワに関係なく，おおむね交互に全部取込をしているような特徴が見られる．グループ D とグループ E のインタラクション図の一部をそれぞれ**図 9** と**図 10** に示す．図 10 に関しては完全に交互に全部取込をしている様子が見られ，図 9 に関してもおおむね交互に全部取込をしている．利用群の 2 人グループ

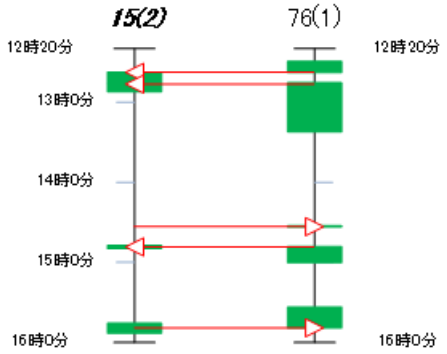

図 9　グループ D のインタラクション図の一部

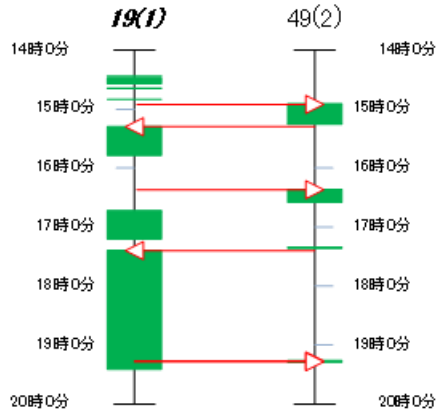

図 10　グループ E のインタラクション図の一部

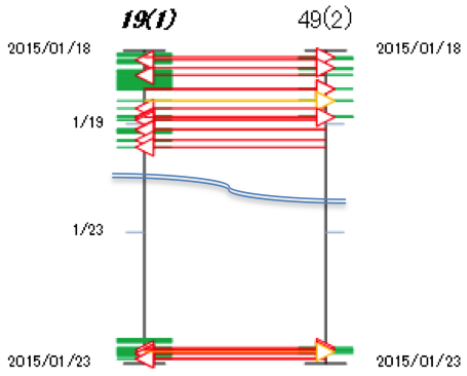

図 11　グループ E のインタラクション図（取込期間）

7 組中 6 組に関して，インタラクション図の一部にこのようなパターンが見られる．

2 人グループは，交互に全部取込を行うという特徴があった．このことから，ドライバとフォロワに関係なくお互いがプログラムを拡張していると考えられる．図 10 のグループ E の例では，まず 49 が 19 の最新のものを取り込んで，取り込んだものを拡張して，拡張したものを 19 が再度取り込んで拡張するという流れでプロジェクトが進んでいると想定できる．

6.2.4　取り込み期間

利用群全 15 組の内，12 組は最初の取込は課題開始から 1 週間後以降であり，さらにその内の 11 組は締め切り日

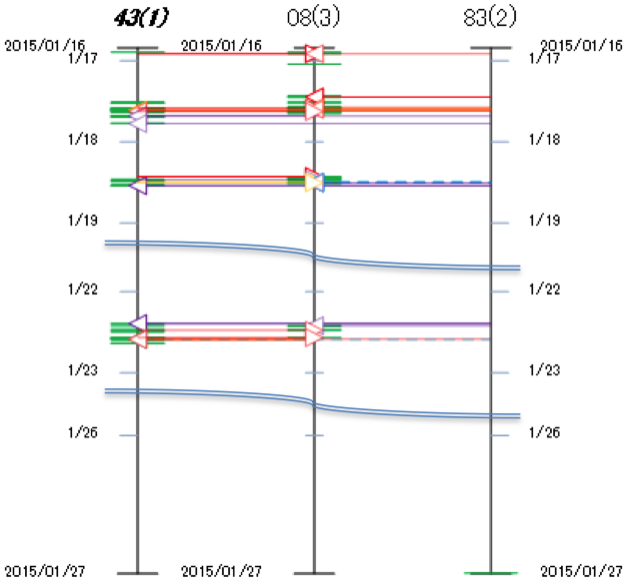

図 12 グループ H のインタラクション図（取込期間）

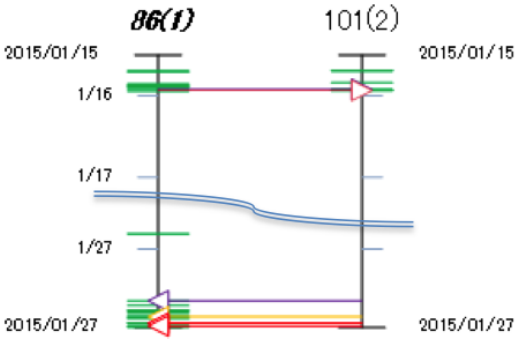

図 13 グループ J のインタラクション図（取込期間）

前夜以降に取り込み操作が見られた．しかし，3 組のみ課題開始日もしくはその翌日から取り込み操作が見られる．その 3 組のインタラクション図をそれぞれ**図 11**，**図 12**，**図 13** に示す．最初の取り込みが早い 3 グループ（早期着手グループ）の特徴として，最後の取り込みも早々に行われていることが分かる．いずれのグループも締め切り前日には取り込み操作が見られない．特にグループ E とグループ H に関して，最後の取り込みは締め切り日よりも 1 週間近く前に行われている．

15 組中 12 組が，最終課題開始日から 1 週間後以降に取り込み操作が初めて行われている．この理由として，最終課題開始日には別のプログラムを組む課題が課されていたことが考えられる．開始日から 1 週間後の講義では課題はなく，講義時間も最終課題の準備時間として提供された．このことから，開始日から 1 週間後以降に取り込み操作を行うグループが大半だったと考えられる．

早期着手グループの中で，最初の取り込みと最後の取り込みの時期が特に早かった 2 組の成績と作品の総ステップ数を，利用群の平均と比較したものを**表 6** に示す．表 6 より，早期着手グループのメンバーの成績はいずれも平均よりも高い．このことから，最終課題までの講義の課題が円滑に完了していたために着手が早かったと考えられる．成績が高いながら，作品の総ステップ数としては利用群の平均値と大きな差がないことと取り込み期間が短かったことから，作業時間も短かったと考えられる．

表 6 早期着手グループの成績と作品

グループ ID	個人 ID	成績	作品総ステップ数
E	19	55	320
	49	52	
H	08	54.5	472
	43	59	
	83	58	
利用群平均		51.2	460

表 7 協調プログラミングにおけるフォロワの遠慮

	利用群	非利用群
遠慮しなかった	7 人（35.0%）	5 人（15.2%）
やや遠慮しなかった	7 人（35.0%）	12 人（36.4%）
やや遠慮した	6 人（30.0%）	12 人（36.4%）
遠慮した	0 人（0.0%）	4 人（12.1%）

表 8 協調プログラミングにおけるフォロワの貢献

	利用群	非利用群
貢献できた	9 人（45.0%）	11 人（33.3%）
やや貢献できた	11 人（55.0%）	14 人（42.4%）
やや貢献できなかった	0 人（0.0%）	7 人（21.2%）
貢献できなかった	0 人（0.0%）	1 人（3.0%）

表 9 グループ L と N のメンバーの成績

グループ ID	個人 ID	成績
L	50	58.5
	71	58.5
	72	51.5
N	18	43.5
	30	54
	31	42.5

6.3 アンケート結果

遠慮と貢献について尋ねたアンケート結果のうち，フォロワのデータを抽出したものをそれぞれ**表 7** と**表 8** に示す．フォロワはグループの 2 番手 3 番手にあたるメンバーであり，グループ内での成績の順位によって決めた．それぞれ有意水準 5%で t 検定にかけたところ，有意差が見られた．

7. 考察

7.1 collective contribution

プログラムのパーツを取り込むようなインタラクションパターン（図 4 や図 5）と，締め切り前日に全部取込を行ってテストを行っていると考えられるインタラクション

パターン（図 5 や図 8）から，フォロワがプロジェクトに貢献をしていることが分かる．

グループ L は，50 のプログラムが 71 に部分取込され，72 が 71 のプロジェクトを全部取込してテストをしていると考えられる．グループ L と N の各メンバーの成績を**表 9** に示す．50 と 71 の成績は同じであり，72 のみ成績に差があるため，このようなインタラクションパターンになったと考えられる．グループ N について，フォロワが全部取込を行いテストをしていると考えられる．表 9 より，ドライバとフォロワの成績の差が大きいため，フォロワはテストという点で貢献したと考えられる．

以上のとおり，能力が一律ではないグループメンバーが実力に応じて貢献していると考えられる．

7.2 productive interaction

2 人グループにおいて，6 組が交互に全部取込するパターンが見られたことから，他人のプロジェクトを取り込み，ソースコードを読んだ上で拡張していることが考えられる．

3 人グループにおいては，グループ K のインタラクション（図 4）では，フォロワがドライバのプロジェクトを全部取込した後にドライバがフォロワのソースコードを部分取込していることから，フォロワはドライバのプロジェクトについて何らかの拡張をし，出来上がったパーツをドライバが取り込んでいると考えられる．

以上のとおり，被験者はグループメンバーのソースコードを読んで拡張を行っていると考えることができる．

8. おわりに

プログラミング入門教育の場で，グループでプログラムを組む機会が設けられている．初学者が協調プログラミングを行うことが困難となっている問題を解決するために，独立同期モデルを提案し，協調プログラミング支援システムを開発し，本稿ではシステムのログを基にグループメンバー同士のインタラクションの分析を行った．

プログラミング入門講義においてシステムを導入し，そのログから初学者の協調プログラミングにおいて，4 つのインタラクションパターンが見られた．その中で協調プログラミングの定義を満たすと考えられるパターンも見られた．アンケート結果から，利用群と非利用群を比較したところ，フォロワの遠慮の減少と貢献の増加に有意差が見られた．

謝辞 本研究は JSPS 科研費 25730203，26280129 の助成を受けたものです．

参考文献

[1] 松浦佐江子，相場亮：グループワークによるソフトウェア工学教育の試み，情報処理学会研究報告―コンピュータと教育，第 2003 巻，pp.1-8 (2003).

[2] 玉田春昭，井垣宏，引地一将，門田 暁人，松本 健一：プログラミング実習におけるグループ開発プロセスの分析，ソフトウェア工学の基礎 XII，日本ソフトウェア科学会 FOSE2005，pp.123-128 (2005).

[3] Schwaber, K., & Beedle, M.: *Agile software development with scrum*, Series in agile software development, Prentice Hall (2002).

[4] Takeuchi, H., & Nonaka, I.: *The New New Product Development Game*, Harvard Business Review, JANUARY-FEBRUARY 1986, pp.137-146 (1986).

[5] Kidane, Y. H., & Gloor, P. A.: *Correlating temporal communication patterns of the Eclipse open source community with performance and creativity*, Computational and mathematical organization theory, 13(1), pp.17-27 (2007).

[6] Knutas, A., Ikonen, J., & Porras, J.: *Communication Patterns in Collaborative Software Engineering Courses: A Case for Computer Supported Collaboration*, Koli Calling ’13 Proceedings of the 13th Koli Calling International Conference on Computing Education Research, pp.169-177 (2013).

[7] 平井佑樹：協調プログラミング学習支援のための分散および対面協調学習の分析，筑波大学博士（情報学）学位論文 (2012).

[8] Beck, K.: *Extreme Programming Explained: Embrace Change*, Addison-Wesley (1999).

[9] Cockburn, A., & Williams, L.: *The Costs and Benefits of Pair Programming*, in First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2000), pp.223-243 (2001).

[10] Goldman, M., Little, G., & Miller, R. C.: *Real-time collaborative coding in a web IDE*, In Proceedings of the 24th annual ACM Symposium on User Interface Software and Technology (UIST), ACM, pp.155-164 (2011).

[11] Vandeventer, J., & Barbour, B.: *CodeWave: A Real-Time, Collaborative IDE for Enhanced Learning in Computer Science*, In Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE ’12), ACM, pp.75-80 (2012).

[12] Salinger, S., Oezbek, C., Beecher, K., & Schenk, J.: *Saros: An Eclipse Plug- in for Distributed Party Programming*, In Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Soft-ware Engineering, pp.48-55 (2010).

[13] Brodahl, C., Hadjerrouit, S., & Hansen, N. K.: *Collaborative Writing with Web 2.0 Technologies: Education Students’ Perceptions*, Journal of Information Technology Education: Innovations in Practice, Vol. 10, pp.73-103 (2011).

[14] Zhou, W., Simpson, E., & Domizi, D. P.: *Google Docs in an Out-of-Class Collaborative Writing Activity*, International Journal of Teaching Learning in Higher Education, 24, pp.359-375 (2012).

[15] André, P., Kraut R. E., & Kittur, A.: *Effects of Simultaneous and Sequential Work Structures on Distributed Collaborative Interdependent Tasks*, In CHI’14, pp.139-148 (2014).

[16] 加藤優哉，松澤芳昭，酒井三四郎：独立同期モデルによる初学者向け協調プログラミング支援システムの開発，情報処理学会情報教育シンポジウム（SSS）2014，pp.27-34 (2014).