情報処理学会論文誌　教育とコンピュータ　Vol.3　No.2　28–40 (June 2017)

研究会推薦論文

CheCoPro：協調的知識創造を指向した初学者の協調プログラミング支援システム

加藤 優哉[1,a)]　松澤 芳昭[1,†1,b)]　酒井 三四郎[1,c)]

受付日 2015年12月18日，再受付日 2016年4月21日 / 2017年1月4日，
採録日 2017年3月4日

概要：本研究では，初学者の協調プログラミングを支援するシステム「CheCoPro」の提案と評価を行った．協調的知識創造の理論と教育実践の問題点の整理を通して，「協調プログラミング」を，(1) **collective contribution**：能力が一律ではないグループのメンバ個々人が能力に見合った貢献ができていること，(2) **productive interaction**：グループメンバのインタラクションによって，各メンバの知識・能力が相乗的にプロジェクトに反映されていること，と定義した．初学者の協調プログラミング支援のために (a) 独立した個々のブランチを管理する，(b) グループメンバの最新バージョンをリアルタイムに閲覧できる，(c) グループメンバのソースコードを単純操作で取り込むことができる，という 3 点の特長を持った「独立同期モデル」を提案し，提案モデルに基づくソフトウェア「CheCoPro」を開発した．文科系の学生 100 名を対象とする大学 1 年次のプログラミング入門教育で実践を行った．システムに記録された操作ログを用い，グループメンバ間相互作用を表すインタラクション図を作成し，質的分析を行った．その結果，協調プログラミングの 2 つの要件を充足すると解釈されるインタラクションパターンが見られた．アンケート結果も，グループメンバ間の能力差にかかわらず，個々のメンバが遠慮せず，より満足度の高い貢献を行うことができたことを示した．

キーワード：協調プログラミング，プログラミング教育，CSCL，初学者，インタラクション

CheCoPro: A Software Promoting Collaborative Programming for Novices through Collaborative Knowledge Creation Approach

Yuya Kato[1,a)]　Yoshiaki Matsuzawa[1,†1,b)]　Sanshiro Sakai[1,c)]

Received: December 18, 2015, Revised: April 21, 2016/January 4, 2017,
Accepted: March 4, 2017

Abstract: We have developed “CheCoPro”: a software to support collaborative programming in introductory programming education. Through literature review and reflection on our experiences, we defined collaborative programming as it fulfils the following two requirements: (1) **collective contribution**: all members make contributions to the group products on an assumption of skill level differences between members, and (2) **productive interaction**: a product is not a result of working individually, but the procedure includes activities of reading others’ code, improvement, and expansion. To support the collaborative programming, we proposed the model named “individual work branch and real-time sharing”, which has the following features: (a) changes made by someone do not directly affect others’ code, (b) it is always possible to view others’ latest programs, and (c) novice programmers can import others’ programs in a simple way. We implemented the model as a workable software, and it was examined using 100 liberal arts students in our introductory programming course. Qualitative analysis was conducted with the interaction charts which is created by using activity logs in the system and it visualizes an interaction pattern representing a trajectory in interaction of collaborative programming. The results showed that the visualized patterns were interpreted to fulfil two requirements of the defined collaborative programming. The results of questionnaire by students showed that despite of big differences in skills between group members, the system reduced refraining to contribute a group product, and it led a high achievement to their contribution.

Keywords: collaborative programming, programming education, CSCL, novice, interaction

1. はじめに

ICT を活用しながら学習者同士が互いに知識・経験・技能を豊かにし，問題解決の方法を見出し，協調的なプロセスを通して新しい知識を創造する能力が 21 世紀に必要な能力とされている [1]．ソフトウェア開発の現場においても，近年では協調的で創造的なソフトウェア開発手法のニーズが高まっている．そのような背景でアジャイルソフトウェア開発手法，特に，知識創造理論 [2] をベースとした Scrum [3] が注目を集めており，教育現場でもその導入が試みられ始めている [4]．

プログラミング入門教育においても，グループでプログラムを作成する課題が試みられてきた [5], [6]．しかしながら，実際の現場では，初学者がグループプログラミングを通して，協調的な創造活動を行うことは困難である．我々の現場の観察では，典型例として，(1) プログラムの大半をグループで最も得意な学生が記述してしまい，その他のメンバはプログラムを書かない，(2) グループメンバが完全にタスクを分割して作業をしてしまう（たとえば，複数の小規模なサブゲームからなるゲームをそれぞれ完全に独立して制作する）などという問題が観察される（2.2 節で詳説）．

初学者がグループプログラミングで使用するツールにも問題がある．理想としては，構成管理ツール（e.g. subversion，git，mercurial）を駆使してプロジェクトを進めることが望ましい．しかしながら，プロフェッショナル向けにデザインされているためツールの操作や概念が複雑で，プログラミング入門講義の学習者が使用するのは困難である．プログラミング入門教育で学習者が集中すべきなのはアルゴリズム構築であり，このようなツールの利用方法に関して学ぶ時間は用意されていない．

本研究では，これらの問題を解決するために，初学者向けの協調プログラミング支援システム「CheCoPro」を提案する．初学者が容易に利用でき，かつ能力差があるグループでも，個々人が並行的に作業し，貢献できる構成管理ツールのモデルを考案し，実際の授業で試行した．システムの利用ログを利用することで初学者のグループプログラミングにおけるインタラクションを視覚化することで，インタラクションの実態解明とあわせてツールの評価を行った．

本論文は全 9 章からなる．プログラミング教育における「協調プログラミング」の定義を 2 章で行う．3 章で，グループプログラミングを支援する既存ツールのレビューを行う．4 章では，初学者に協調プログラミングを支援するツールのモデル「独立同期モデル」を提案する．5 章では，4 章で提案したモデルに基づいて設計・実装したシステムを説明する．6 章で評価実験の方法，7 章ではその結果を報告する．8 章で考察を行う．9 章はまとめである．

[1] 静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University, Hamamatsu, Shizuoka 432–8011, Japan
[†1] 現在，青山学院大学社会情報学部
Presently with School of Social Informatics, Aoyama Gakuin University
[a)] kato@sakailab.info
[b)] matsuzawa@si.aoyama.ac.jp
[c)] sakai@inf.shizuoka.ac.jp

2. 協調プログラミング

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．なかでも，ウォータフォールモデルとアジャイルモデルの 2 つのモデルが主要である．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 [2] がベースとなっている．したがって，ウォーターフォールモデルとは正反対の協調作業が行われる．たとえば，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あげられる．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 [7]．教育現場でも同様の現象が観察されている [8]．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習者間のインタラクションの重要性が主張されている．平井は，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，合意形成などを繰り返し，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 [9]．

ソフトウェア開発の現場の観点から，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）[10] 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 [11]．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 [12]．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　入門環境における協調プログラミングの現状

我々は，プログラミング入門授業の場で，最終課題として 1 チーム 2，3 名から構成されるグループプログラミングを実施してきた．現状調査のために，2013 年度の授業でアンケートと観察による協調プログラミングの調査を行った．約 100 名の受講者に対するアンケートを行い，70 件の有効回答を得た．

我々が最も注目したのは，グループメンバ間の能力差についてである．まず，各グループのメンバについて次の 2 つの役割に分類した．

ドライバ　グループ内で，コーディングにおいてプロジェクトに最も主体的に参加しているメンバ

フォロワ　ドライバ以外のメンバ

ドライバとフォロワの実際の記述量に関するアンケートの結果，グループ内で最もプログラミングに対する得意意識が高い学習者は，成果物のソースコードの記述量の平均が 67% [*1] であることが分かった．その他のメンバのソースコード記述量は，平均 28%であり，まったく記述していない学習者も存在した．

観察結果から，2 つの典型的な問題があることも分かった．

1 つ目の問題は，個々の能力に見合わない不適切な役割分担である．我々は，プロジェクトに費やす時間やソースコードの記述量が，グループメンバ同士で等しくなるように課題にとりかかることは意図していない．しかし，学習者は各グループメンバの責任を平等にするために，作業量が等しく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，プロジェクトが完全に独立した作業からなることである．たとえば 3 人グループのプロジェクトにおいて，メンバそれぞれが 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な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 3 つのプロジェクトを個別に開発したのみであり，協調的な作業とはいえない．

2.3　協調プログラミングの定義

我々が目指す協調プログラミングは，2.2 節で述べた 2 つの問題を解決し，グループメンバ同士のインタラクションによって，創発的に創造的な成果を生み出すような活動が行われるものである．本研究における協調プログラミングは，以下の 2 つの要件を満たすものとする．

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

productive interaction　グループメンバのインタラクションによって，各メンバの知識・能力が相乗的にプロジェクトに反映されていること．

グループでのプログラミングでは，単にコーディングだけではなく様々な場で貢献することが可能である．各メンバが自身の能力を活かすことができる場で貢献することを「能力に見合った貢献」とする．collective contribution の達成は，フォロワの遠慮を取り除くことで可能であると考えられる．フォロワの遠慮は，ドライバに最新のプロジェクトを送信するように要求するときや，ドライバのソースコードを編集するときに発生することがある．フォロワの遠慮を取り除くことで，フォロワが積極的にプロジェクトに参加可能となる．結果，能力に見合った貢献が可能となると考えられる．

グループでのプログラミングでは，メンバ間でインタラクションがあることが望ましい．インタラクションによって新しい知識が生じ，それによって制作物が改善されて品質が高くなることを「各メンバの知識・能力が相乗的にプロジェクトに反映されていること」とする．productive interaction の達成は，フォロワの貢献度合いに依存するものと考えられる．ドライバよりもプログラミングが苦手とされるフォロワの貢献は，ドライバや他のフォロワとのインタラクションから発生すると考えられる．

3.　先行研究

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 [12]．Collabode は，複数の学生が同時にソース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 [13]．技術的には，一般的な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 [14]．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

[*1] 学習者の主観による

ていない．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かかわらず，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．一般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 [15], [16]．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とりかかることには不利であると主張している [17]．一方で，プログラミングの入門教育において，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あげられる [16]．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，1 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．

4. 独立同期モデルの提案

本章では，既存のコラボレーション支援ツールをモデル化し，その特徴を述べた後，本研究で初学者が利用できるように考案したモデルを提示する．既存のモデルを 2 つにモデル化し (a) SCM 共有モデル，(b) リアルタイム共有モデルとし，我々が考案した (c) 独立同期モデルとともに，**図 1** に示す．

4.1 (a) SCM 共有モデル

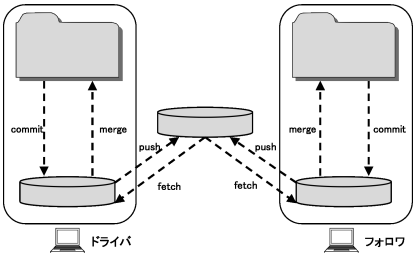
SCM ツールの主な機能は版管理，共有，構成管理である．本章では，コラボレーションの支援に着目して議論するため，SCM ツールの主な機能のなかでも共有に着目してモデル化を行った．

SCM 共有モデルは，git のようなソースコードの変更履歴を記録・追跡するための分散型構成管理ツールのモデルである．SCM 共有モデルを図 1 (a) に示す．図のフォルダアイコンはプロジェクトフォルダを表す．実線矢印は同期，破線矢印は取り込みを表す（ほか 2 つのモデルも同様である）．SCM 共有モデルは以下の特徴を持つ．

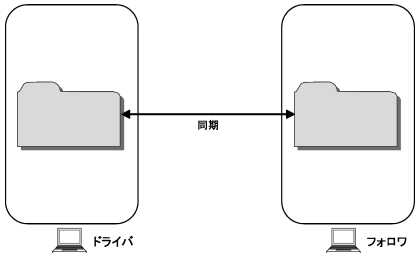
(1) 1 人で複数のブランチを管理することができる．
(2) push・fetch 操作により最新バージョンを共有する．
(3) 取り込みは marge（差分取込）を用いる．

SCM 共有モデルには 2 つの問題点がある．

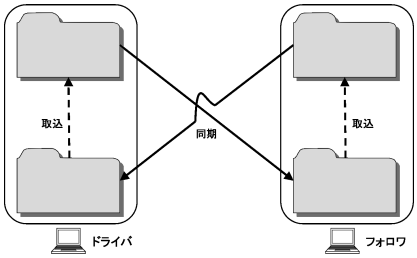
1 つ目は，push や fetch，merge といった様々な専門用語の概念や操作方法が初心者にとって複雑な点である．プログラミング入門教育においては，ほかに重要な学習項目



(a) SCM 共有モデル



(b) リアルタイム共有モデル



(c) 独立同期モデル

図 1　協調プログラミング支援システムのモデル比較
Fig. 1　Comparison of models of collaborative programming support system.

が多数あるため，このような不必要な学習負荷は避けるべきである．

2つ目は，ドライバが最新のプロジェクトを push しなければ，フォロワがドライバの最新のプロジェクトを入手することができない点である．このような状況の際に，フォロワはドライバに最新のプロジェクトを push するように要求する必要がある．しかし，フォロワはドライバの作業を阻害してしまうことを恐れて要求できないことがある．

4.2　(b) リアルタイム共有モデル

リアルタイム共有モデルは，Saros のようなリアルタイムにファイルを同期し，多人数で同時に同一ファイルへの書き込みが可能なツールのモデルである．リアルタイム共有モデルを図 1 (b) に示す．リアルタイム共有モデルは以下の特徴を持つ．

(1) グループ内のすべてのメンバが 1 つのプロジェクトを共有する．
(2) 共有されたプロジェクトはリアルタイムに更新・表示される．
(3) すべてのメンバが共有されたプロジェクトを同時に編集できる．

リアルタイム共有モデルの問題点は，ソースコードを編集することがフォロワにとって困難である点である．このモデルは，個人の編集がグループで共有しているファイルに直接変更を加えるため，ドライバが制作したプログラムをフォロワが壊してしまうということが起こりうる．したがって，ソースコードの編集には注意が必要となり，フォロワがプロジェクトに参加しにくい．

4.3　(c) 独立同期モデル

独立同期モデルは，プログラミング初心者の協調プログラミングを促進するために，我々が提案するモデルである．独立同期モデルを図 1 (c) に示す．独立同期モデルは以下の特長を持つ．

(1) 独立した個々のブランチを管理する．
(2) グループメンバの最新バージョンをリアルタイムに閲覧できる．
(3) グループメンバのソースコードを単純操作で取り込むことができる．

特長 (1) によって，複数のブランチを管理するような SCM 共有モデルの複雑さが解消される．グループのプロジェクトに直接干渉してしまうというリアルタイム共有モデルの問題点も解消されると考えられる．この特徴によって，フォロワはドライバのプロジェクトを直接編集してしまうような心配がなくなり，積極的にプロジェクトに貢献ができるようになると考えられる．

特長 (2) はリアルタイム共有モデルの利点を採用し，SCM 共有モデルの問題点を解決している．グループメンバに最新バージョンの push 要求を行う必要がなく，他のメンバの作業を中断する心配を解消することができる．この特長によって，フォロワはドライバへのファイル送信要求をする必要がなくなる．結果，フォロワの遠慮が減少することが予想できる．

特長 (3) によって，ツールの操作方法などを学ぶ学習負荷を解消することができる．取り込みが容易であれば取り込み回数も増え，他のメンバのソースコードの閲覧・編集が行われる可能性がある．これは，productive interaction の手助けになると考えられる．

4.1 節冒頭において，SCM 共有モデルに関して共有機能部分のみモデル化を行うと述べた．独立同期モデルでは版管理機能をサポートしていない．今後，独立同期モデルでも版管理機能をサポートすることを検討している．

5.　CheCoPro

独立同期モデルに基づく協調プログラミングの支援システム CheCoPro（Cheerful Collaborative Programming）の設計・実装を行った．CheCoPro 開発の主目的は，初学者でも利用可能なインタフェースの提供と環境設定の簡略化により，初学者が，独立同期モデルに基づく協調プログラミングを確実に実施できる環境を提供することである．CheCoPro はクライアント・サーバ方式を利用したシステムである．サーバは新規開発し，クライアントは，我々が授業で利用しているプログラミング初学者用の開発環境上に組み込む形で開発した．開発言語はクライアント・サーバともに Java（version 1.8）である．

近年では，Dropbox などのリアルタイムにファイルを共有できるツールをうまく用いることでも，独立同期モデルに基づく協調プログラミングの運用は可能ではある．しかし，その場合には，メンバ個々人が，書き込み権限が自身にしかない共有フォルダを作成し，他メンバのファイルは閲覧のみ可能な環境設定を行う，もしくはそれに準ずる運用ルールを策定し，厳密に運用する必要がある．初学者がそうした環境で独立同期モデルを確実に運用することは難しく，リアルタイム共有モデルに近い運用をしてしまうことが危惧される．CheCoPro は，ツールで操作に制約をかけることによって，初学者が独立同期モデルを理解していなくても，自然に当モデルに基づく運用ができるように設計されている．本章では，その CheCoPro の特長である独立同期モデル支援機能について述べる．

5.1　インタフェース

CheCoPro によるファイル共有の概略図を**図 2** に示す．エディタは自身のプログラムを記述するためのウィンドウである．ビューアはメンバのプロジェクトを閲覧するためのウィンドウである．ビューアは，メンバセレクタ（**図 3**）上のメンバの ID が表示されたボタンをクリックすること

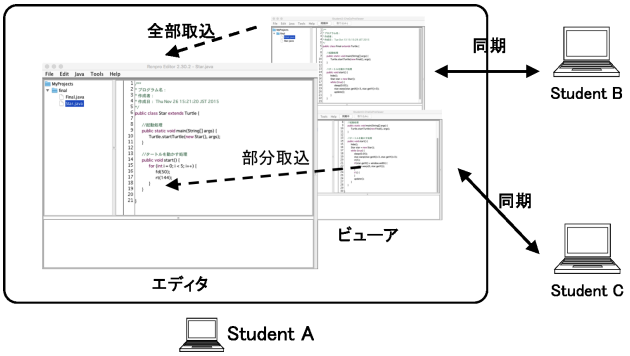


図 2　CheCoPro によるファイル共有の概略図
Fig. 2　Schema of file sharing by CheCoPro.

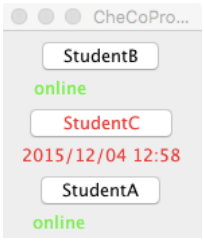


図 3　メンバセレクタ
Fig. 3　Member selector.

図 4　メニューバー
Fig. 4　Menubar.

で開く．メンバセレクタ上に黒字で表示されているメンバはオンラインであり，赤字で表示されているメンバはオフラインである．ビューアのメニューバー（図 4）から同期状態にするか否かや，全部取込操作を行うことができる．

以下，同期機能と取り込み機能の詳細を説明する．

5.2　同期機能

同期機能は，グループメンバの最新のプロジェクトをリアルタイムに受け取り，それを観察できる機能である．ユーザが自身のエディタ上でソースコードの編集を行うと，その様子が他のメンバのビューアにリアルタイムに反映される．ビューアでは，編集を観察するだけでなく，プログラムを実行して動作を確認することも可能である．

ビューアではソースコードの編集ができない．つまり，他のメンバのソースコードを直接変更できない仕様である．これは独立同期モデルに基づいた設計である．他のメンバのソースコードを編集したい場合，編集したいソースコードを取り込み自分の変更を加え，他のメンバが取り込むという手順が必要である．

ビューアのメニューバーから同期/非同期の切替えを可能とした理由は，ビューアからメンバのプログラムを実行可能にするためである．同期中はメンバの変更をリアルタイムに受け取るため，プログラムをコンパイルし，実行することができない．非同期中に切替え，コンパイルエラーが発生するような状態なら自身で修正をして実行することができる．

5.3　取り込み機能

取り込み機能とは，単純な操作でグループメンバのファイルを取り込むことができる機能である．CheCoPro では，2 種類の単純な取り込み方法を提供している．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，1 人のグループメンバが持っているファイルを 1 度にすべて取り込むというものである．全部取込を行う際に，プロジェクトの中のファイルをすべて取り込むか，ソースファイル（Java ファイル）を取り込むか，素材ファイル（画像や音楽ファイル）を取り込むかを選択することが可能である．取り込む際に，自身のプロジェクトに同名ファイルが存在する場合は上書きされる．

部分取込は，コピー&ペーストによる手動の取り込み方法である．ユーザはビューアから取り込みたい部分をコピーし，エディタ上に貼り付けるという単純操作でソースコードを取り込むことができる．

以上の 2 点の操作には特別な知識は必要としないため，初心者でも容易に行うことができると考えられる．

6.　実験方法

CheCoPro の協調プログラミング支援効果の検証を目的に実験を行った．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，アンケートによる協調プログラミングの意識調査を行った．

6.1　学習環境と課題内容

2014 年度秋学期に開講された静岡大学のプログラミング入門講義において実験を行った．プログラミング入門講義は，週 2 コマの必修の授業である．受講者は，情報学部情報社会学科（文科系）1 年生である．実験を行うまでに，受講者はオブジェクト指向の概念を除く Java の基礎的な学習に 4 カ月間取り組んでいる．協調プログラミングに取り組むことは未経験である．

実験は，当該講義の最終課題として行った．最終課題の要項を**表 1** に示す．最終課題の内容は，Java を用いた 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．グループは上限 3 名であり，メンバは受講者が任意で決めたものである．

6.2　システムの導入と利用群，非利用群のグループ分け

まず，最終課題にエントリした全 44 組（104 名）から，1 人グループ 8 組を除いた 36 組（96 名）を分析対象とした．

CheCoPro の導入方法と機能に関する説明を行ったうえ

表 1　課題内容
Table 1　The final project description.

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

表 2　グループの内訳と成績
Table 2　Number of groups and records of both groups.

	利用群	非利用群
組数（割合）	15 組（41.7%）	21 組（58.3%）
成績の平均*2	51.2 (sd=7.38)	49.4 (sd=8.99)

で，CheCoPro を利用するかどうかは任意とし，最終的にそのグループの利用状況で利用群と非利用群を分けた．その結果を**表 2** に示す．利用群は 15 組（3 人グループ 8 組，2 人グループ 7 組），非利用群は 21 組（3 人グループ 16 組，2 人グループ 5 組）であった．

このとき，CheCoPro の導入方法と機能に関する説明は，被験者全員に対して行った（約 10 分）．その後，全グループに対して，CheCoPro がすぐに使えるよう初期設定を行った．このようにして，初期設定の煩わしさを理由として CheCoPro 非利用を学習者が選択することを避けた．

この方式での利用，非利用群の分離は，学習者が学習の状況を考えて環境を学習者自身が選択できる反面，利用群と非利用群にプログラミングの能力差が出る可能性がある．そこで，各群のプログラミング能力がおおむね等質であることを調べるために，各群の成績の差を調べた．表 2 に示された成績は最終課題に入る前までの講義の成績であり，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．利用群の平均は 51.2（sd=7.38），非利用群の平均は 49.4（sd=8.99）であった．t 検定（以下，断りのない限り Welch の法による）を行った結果，平均の差は有意でなかった（両側検定：t(89.0)=1.10, n.s.）．以上より，両群のプログラミング能力について同等であることを確認した．

6.3　分析方法
6.3.1　アンケート
協調プログラミングに対するフォロワの意識調査から，独立同期モデルの有効性を検証するために，アンケート結果の分析を行った．

アンケート項目の中で，協調プログラミングの達成に重要と考えられる「遠慮」と「貢献」について尋ねたアンケート結果に関して分析を行った．協調プログラミング中の遠慮と貢献の度合いを，「強くそう思う」「そう思う」「そう思わない」「強くそう思わない」の 4 段階のリッカート尺度を用いて自己評価を行うアンケートを行い，それらの結果を間隔尺度と解釈して統計処理を行った．

*2　この時点までに取得できる成績の最高点は 73 である．

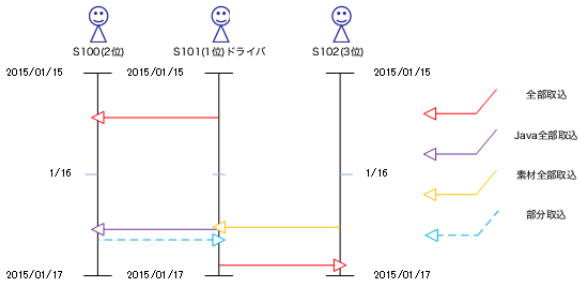


図 5　インタラクション図（例）
Fig. 5　The interaction chart (An example).

フォロワの意識の変化が協調プログラミングの成功につながると考えられるため，分析対象は各グループのフォロワとした．ドライバはコーディングにおいてプロジェクトに最も主体的に参加しているメンバと定義したとおり，プロジェクトに積極的に貢献していることが前提である．しかし，フォロワの貢献度合いに関してはグループごとに異なっていたためである．

6.3.2　インタラクションパターン
プログラミング初学者の協調プログラミングの実態解明を目的に，インタラクション図の作成と質的分析による評価を行った．

インタラクション図とは，グループメンバ同士の取り込みのやりとりを時系列に表した図である．インタラクション図の例を**図 5** 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．各矢印は取り込みを表し，矢印先のメンバが矢印元のメンバのソースコードなどを取り込んだことを表す．

ドライバの抽出については，被験者の成績とインタラクション図を用いて以下の手順で行った．(1) 各グループについて，成績 1 位のメンバを抽出する．(2) 全部取込（素材全部取込を除く）によって取り込まれた回数が最も多いメンバを抽出する．(3) (1) と (2) によって抽出したメンバが一致した場合そのメンバをドライバとする．(4) (3) によってドライバが決定しなかった場合，部分取込をした回数が最も多いメンバと (1) または (2) によって抽出したメンバと一致するメンバをドライバとする．(5) (4) によってドライバが決定しなかった場合，(1) で抽出したメンバをドライバとする．

インタラクション図は利用群全 15 グループに対して作成した．インタラクション図に対する質的な分析から典型的なインタラクションのパターンを抽出することを試みた．

インタラクション図は，ファイルやソースコードの取り込みを表す図である．したがって，被験者がどのような意図で取り込みを行ったのかは把握できない．被験者の行動

の意図については，取り込みを行った日時，頻度，ファイルの種類など，インタラクション図の文脈からの推測となる．推測は被験者に対するインタビューの経験 2 例からおおむね正確であることが分かっている [18].

6.3.3　インタラクションとコラボレーション

協調プログラミングが行われていることを確認するために，メンバの成績と取り込み回数という指標を用いてインタラクション図の分析を行った．

インタラクションパターンから，ソースコードの提供による貢献（7.2.1 項）とテストによる貢献（7.2.2 項）が発生しているグループを抽出し，各メンバの成績と照らしあわせることによって分析を進めた．

7.　実験結果

7.1　アンケート結果

フォロワのアンケートの有効回答数は，53 件（利用群 20 件，非利用群 33 件）であった．

「グループでプログラムを組むときに他のメンバに対して遠慮しましたか」と尋ねたアンケートの結果を**図 6** に示す．図のエラーバーは 95%信頼区間を表す．利用群の平均は 1.95（sd=0.80），非利用群の平均は 2.45（sd=0.89）であった．t 検定を行った結果，利用群と非利用群の平均の差は有意であった（両側検定：t(43.1)=−2.08，p<.05）．

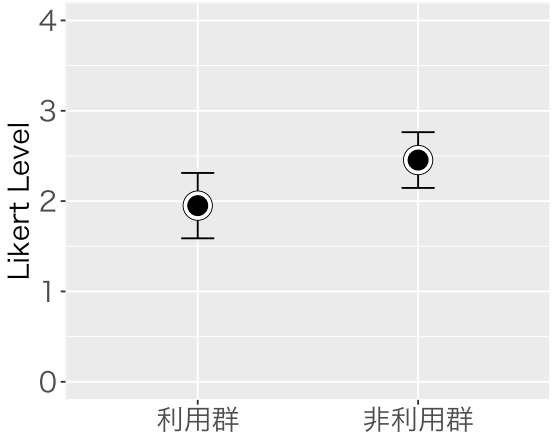


図 6　協調プログラミングにおけるフォロワの遠慮
Fig. 6　Diffidence of folowers in collaborative programming.

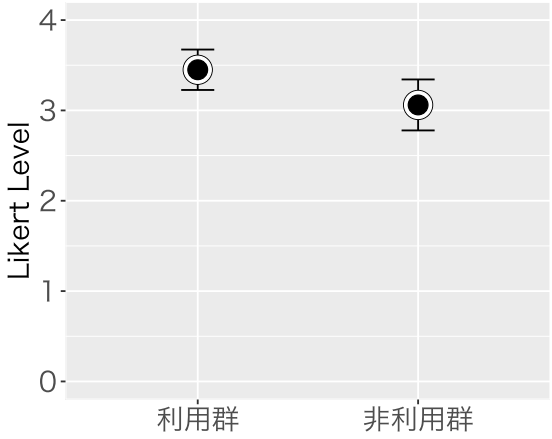


図 7　協調プログラミングにおけるフォロワの貢献
Fig. 7　Contribution of followers in collaborative programming.

「あなたはあなたのグループの作品作りに貢献できましたか」と尋ねたアンケートの結果を**図 7** に示す．図のエラーバーは 95%信頼区間を表す．利用群の平均は 3.45（sd=0.49），非利用群の平均は 3.06（sd=0.81）であった．t 検定を行った結果，利用群と非利用群の平均の差は有意であった（両側検定：t(50.1)=2.12，p<.05）．

「グループでのプログラミングの感想の自由に記述してください」というアンケート項目の結果を**表 3** に示す．S01〜S03 はポジティブな意見を抽出した．ポジティブな意見では，協力することで理解が進んだという旨の意見や，メンバ同士の議論からアイデアを創出できたという意見が見られた．S04〜S06 はネガティブな意見を抽出した．ネガティブな意見では，能力差から貢献ができなかったという意見や，グループでのプログラミングの難しさについて書かれたものが見られた．特に S6 に関して，差分表示・差分取込といった機能を追加することで対応したい．

「CheCoPro についての感想や追加してほしい機能などを自由に記述してください」というアンケート項目の結果を**表 4** に示す．S07〜S09 はポジティブな意見を抽出した．

表 3　グループでのプログラミングの感想
Table 3　Free writing about group programming.

S01	いろんなアイディアが出てくるので，個人で作るよりも楽しく取り組めました．また，自分が分からないことを他のメンバの子が教えてくれたりしたのでよかったです．
S02	「○○をやりたい」という目的が同じでも，人によってやり方が違うのが面白いなと思った．
S03	グループ内での実力さが大きく，自分にできることを探すのに苦労しました．改めてすごいと思いました．このままではだめだと思い，本などを読んで改めて理解したいと思いました．
S04	グループの中で一番プログラミングが苦手だと自負していたので，あまり貢献できていないような気がします．
S05	メンバと予定が合わなくて大変だった．
S06	人の書いたコードを統合させる工程が非常に難しかった．

表 4　CheCoPro についての感想
Table 4　Free writing about CheCoPro.

S07	グループの人がどんなプログラミングをしているのか随時確認できて便利だと思った．
S08	ファイルの受け渡しがとても楽で使いやすかったです．
S09	便利でありがたかったです！ USB などを使うよりも断然手間が省けました．
S10	同じ名前のファイルが上書きされることに気づかず上書きしてしまい少し困りました．
S11	一部だけファイルを取り込みたい．上書きしないようにしてもらいたい．
S12	思いのほかプログラムの共有が難しかった．

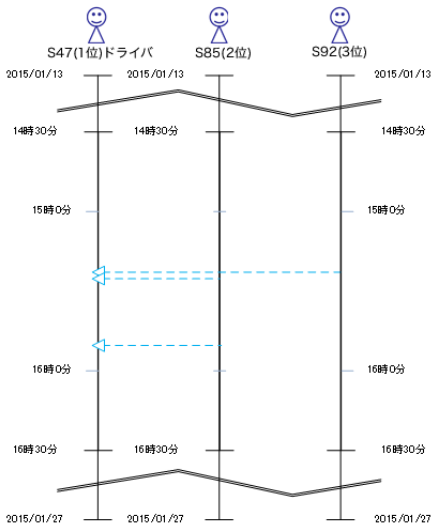


図 8　ソースコードの提供による貢献パターン（Group K）
Fig. 8　Supply of components pattern (Group K).

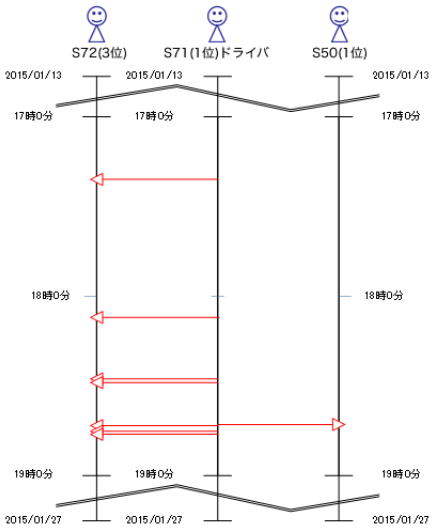


図 9　テストによる貢献パターン（Group L）
Fig. 9　Contribution by testing pattern (Group L).

ポジティブな意見では，メンバの編集を観察できることや取り込みの容易さに関する意見が見られた．S10〜S12 はネガティブな意見を抽出した．ネガティブ意見では，全部取込の際に同名ファイルが上書きされる仕様の改善を求める意見が見られた．

7.2　インタラクションパターン

CheCoPro のログからインタラクション図を作成し，分析を行った結果，3 つのインタラクションパターンが見られた．

7.2.1　ソースコードの提供による貢献

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して採用されることがある．このパターンは全 15 グループ中 7 グループで見られた．

例としてグループ K のインタラクション図の一部を**図 8** に示す．グループ K のフォロワ（S85 と S92）がドライバ（S47）のプロジェクトを全部取込していた．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．

部分取込によって取り込まれたソースコードは，授業の教室を選ぶという，アドベンチャーゲームの 1 シーンである．S47 は S85 から別のシーンに関するソースコードも部分取込している．以上から，グループ K のフォロワはそれぞれ担当するシーンについて責任を担い，実装をしていたと考えられる．

7.2.2　テストによる貢献

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．このパターンは全 15 グループ中 6 グループで見られた．

例としてグループ L のインタラクション図の一部を**図 9** に示す．S72 が S71 から繰り返し全部取込をしていることから，フォロワがドライバのプログラムを繰り返しテストを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．

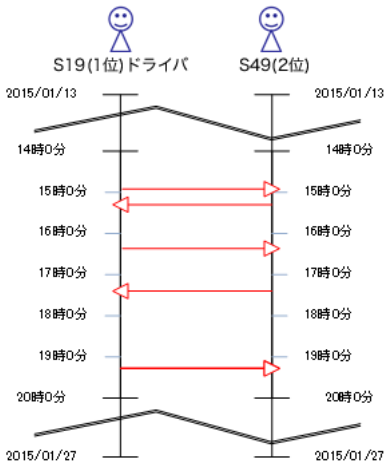


図 10　交互拡張（Group E）
Fig. 10　Alternate extension (Group E).

「あなたは主にどのような作業でグループに貢献しましたか」というアンケートと「あなたは他にどのような作業でグループに貢献しましたか」というアンケートの結果について，フォロワの回答を抽出したものを**図 11** に示す．プログラムの動作テストという項目は最も少ないが，テストで貢献したと答えたフォロワは確かに存在した．

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ことに関して煩わしさが減少し，テストという観点でフォロワがプロジェクトに貢献しやすくなったと考えられる．

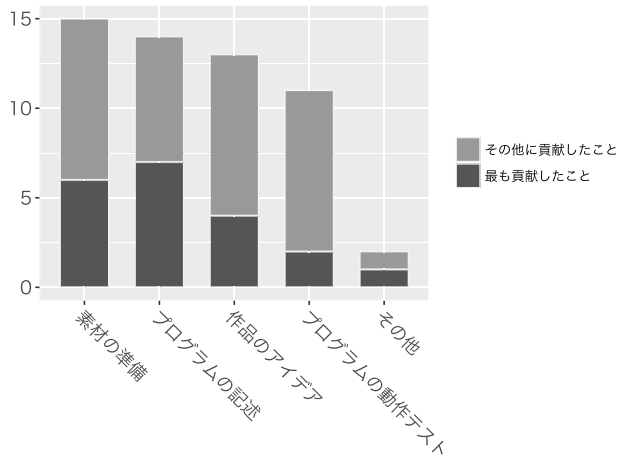


図 11　フォロワが貢献した場面
Fig. 11　When followers have contributed.

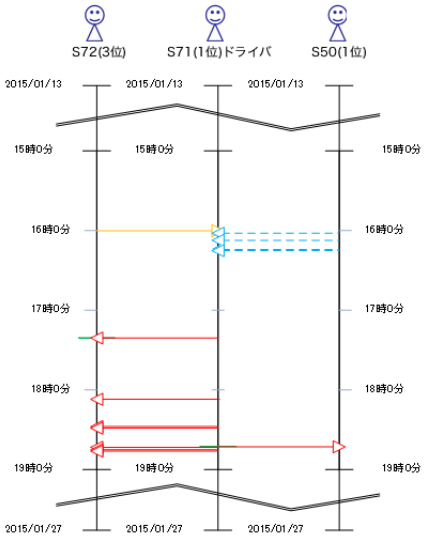


図 12　グループ L のインタラクション図
Fig. 12　The interaction chart of Group L.

7.2.3　交互拡張パターン

グループメンバ同士で交互に全部取込をしている様子が見られる．このパターンは全 15 グループ中 6 グループで見られ，そのすべてが 2 人グループであった．

例としてグループ E のインタラクション図の一部を**図 10** に示す．このパターンからは，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ことが考えられる．グループ E は，まず S49 が S19 の最新のものを取り込んでいる．取り込んだものを拡張して，拡張したものを S19 が再度取り込んで拡張するということを繰り返してプロジェクトが進んでいると想定できる．

7.3　インタラクションとコラボレーション

ソースコードの提供による貢献とテストによる貢献のいずれのパターンも発生しているグループ L のインタラクション図の一部を**図 12**，グループ L の各メンバの成績を**表 5** に示す．S71 は成績 1 位でドライバである．S50 はドライバの S71 と同じ成績であり，インタラクション図からソースコードの一部で貢献していると推測できる．S72 は他のメンバと比べて成績が低く，インタラクション図からテストで貢献していると推測できる．以上のとおり，グループメンバ間に確かに成績差があるなか，それぞれのメンバが貢献できる場で貢献している．

表 5　グループ L の各メンバの成績
Table 5　Records of group L.

ID	成績
S72	51.5
S71	58.5
S50	58.5

表 6　各グループの取込回数
Table 6　Import count of each group.

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

各グループの取り込み回数を**表 6** に示す．表 6 に示したとおり，利用群全グループにおいて取り込み操作が複数回行われている．メンバ間でインタラクションが発生し，ソースコードの提供やテストにおいて貢献できていると考えられる．メンバ間の能力差がさらに大きいグループでは，素材の作成に尽力するメンバが存在する．このような理由から素材全部取込が発生している．ソースコードの提供やテストの促進に加えて，素材の提供という面で貢献するメンバが存在することが明らかになった．

フォロワのテストによって作品が改善されたということは観察からは確認ができなかった．しかし，プログラムの実行ログや，アンケートにおいてプログラムの動作テストで貢献したと答えているフォロワが存在することから，繰り返し動作テストが行われ，それによって作品が改善されたと分析した．

以上のように，成績差があるグループにおいても各メンバが能力に応じてプロジェクトに貢献していることが分かった．

8. 考察

協調プログラミングの 2 つの要件（collective contribution と productive interaction）の観点で考察を行う．2 つの要件の達成度合いから本研究の限界を述べる．

8.1 collective contribution

「能力が一律ではないグループのメンバ個々人が能力に見合った貢献ができていること」を collective contribution の定義とした．特に「能力に見合った貢献」について「各メンバが自身の能力を活かすことができる場で貢献すること」とした．これに対し，能力差が確かにあるなかで，ソースコードの提供，テスト，素材の提供という多様な貢献が行われていることが分かった．以上の成果は，独立同期モデルにおける独立性と取り込みの容易さに起因していると考えられる．

collective contribution の達成においての問題は，責任を平等にするための不適切な役割分担と，ドライバが作業しているプロジェクトをフォロワが編集しにくいことであった．これに対し，フォロワの遠慮を取り除くことが有効であると述べた．アンケートの結果からは，フォロワの遠慮の減少という点で有意差が確認された．以上から，独立同期モデルはフォロワのプロジェクトへの参加を促進する効果があるとも考えられる．

分析を行ううえで，テストや素材の提供で貢献しているメンバが明らかになった．従来の入門教育におけるグループでのプログラミングでは，プログラムの大部分を記述したメンバに評価が偏ることがあった．しかし，作品をよりよくするために，テストの正確性や素材の品質も重要な要素であり，評価するべきである．以上のとおり，コーディング以外の面でプロジェクトの評価が可能となったことも本研究の成果である．

8.2 productive interaction

「グループメンバのインタラクションによって，各メンバの知識・能力が相乗的にプロジェクトに反映されていること」を productive interaction の定義とした．特に「各メンバの知識・能力が相乗的にプロジェクトに反映されていること」について，「インタラクションによって新しい知識が生じ，それによって制作物が改善されて品質が高くなること」とした．フォロワがドライバのソースコード編集の様子を観察していることや，積極的に取り込み操作が行われていることから，各メンバの能力が相乗的にプロジェクトに反映されていると分析をした．

メンバ間のインタラクションによって制作物が改善されて品質が高くなったか否かについては確認ができていない．グループメンバの性格や能力といった様々な要因が影響するため，グループ間で制作物の品質を比較することで productive interaction の達成度合いを調査することはできない．しかしながら，表 6 から分かるように利用群全グループにおいて何らかのインタラクションが発生している．今回，利用群全グループにおいてインタラクションがあったことは，独立同期モデルによる効果があると考えられる．従来，グループワークであるにもかかわらずドライバ 1 人でプロジェクトを進めているグループが存在することが我々の観察で分かっている．以上を考慮し，従来のプログラミング入門教育におけるグループでのプログラミングよりも，CheCoPro を利用したグループの方が productive interaction の達成度が高いと判断した．

観察とアンケートの自由記述からは，知識の伝播や創造的な議論が行われていることが分かった．表 3 の S1 や S2 がその例である．知識の伝播としては，アルゴリズムの構築に関することやコンパイルエラーの解消方法，創造的な議論としては作品のアイデアに関する意見が見られた．フォロワの貢献の増加という点で有意差が見られたことからも，フォロワがプロジェクトに貢献できたという意識が高いことが分かる．以上の成果は，独立同期モデルの特徴である，リアルタイムにメンバの編集の様子が確認できる点と，取り込みの容易性に起因するものと考えられる．

8.3 本研究の限界

独立同期モデルが協調プログラミングの支援に一定の成果をあげたが，いかなるグループに対しても有効に作用するわけではない．アンケートの自由記述において，グループでプログラムを組むことに対してネガティブな意見が見られた．典型的な例は，「プログラミングが苦手であるため貢献できなかった」という意見と，「個人で組むよりも気を遣ってしまい苦労をする」という意見である．

独立同期モデルが有効に働くか否かは，作品の内容，グループ内の能力差や各メンバの性格という様々な要因が重なることによって決まると考えられる．現状では，これらを測定する指標が曖昧なため，分析が困難である．たとえば，プログラミング能力の指標として講義の成績を用いているが，成績がプログラミング能力につながるとは限らない．

支援可能なグループの幅を広げるために，協調プログラミングの実態をより明白にする必要がある．インタラクション図という形でファイルのやりとりを可視化したことで，初学者の協調プログラミングの実態を部分的に明らかにすることができた．協調プログラミング中のコミュニケーションなどのデータも取得し，インタラクション図と組み合わせたより詳細な分析が今後の課題である．

9. まとめ

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ことがある．初学者にとって

協調プログラミングを行うことが困難になっているという問題を解決するために，独立同期モデルを提案し，モデルに基づくシステム CheCoPro を開発した．

CheCoPro をプログラミング入門講義に導入し，協調プログラミングを実施した．CheCoPro の利用ログとアンケートの結果を分析し，独立同期モデルの有効性の検証と協調プログラミングの実態解明を試みた．

その結果，協調プログラミングの 2 つの要件を充足すると解釈されるインタラクションパターンが見られた．アンケート結果も，グループメンバ間の能力差にかかわらず，個々のメンバが遠慮せず，より満足度の高い貢献を行うことができたことを示した．

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] Griffin, P., McGaw, B. and Care, E.: *Assessment and Teaching of 21st Century Skills*, Springer Science+Business Media (2012). 三宅なほみ，益川弘如，望月俊男（訳）：21 世紀型スキル 学びと評価の新たなかたち，北大路書房 (2014).

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

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

[4] Anslow, C. and Maurer, F.: An Experience Report at Teaching a Group Based Agile Software Development Project Course, *Proc. 46th ACM Technical Symposium on Computer Science Education*, pp.500–505 (2015).

[5] 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，Vol.2003, pp.1–8 (2003).

[6] 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).

[7] Kidane, Y.H. and Gloor, P.A.: Correlating temporal communication patterns of the Eclipse open source community with performance and creativity, *Computational and mathematical organization theory*, Vol.13, No.1, pp.17–27 (2007).

[8] Knutas, A., Ikonen, J. and Porras, J.: Communication Patterns in Collaborative Software Engineering Courses: A Case for Computer Supported Collaboration, *Koli Calling '13, Proc. 13th Koli Calling International Conference on Computing Education Research*, pp.169–177 (2013).

[9] 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).

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

[11] Cockburn, A. and Williams, L.: The Costs and Benefits of Pair Programming, *1st International Conference on Extreme Programming and Flexible Processes in Software Engineering* (*XP2000*), pp.223–243 (2001).

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

[13] Vandeventer, J. and Barbour, B.: CodeWave: A Real-Time, Collaborative IDE for Enhanced Learning in Computer Science, *Proc. 43rd ACM technical symposium on Computer Science Education* (*SIGCSE '12*), ACM, pp.75–80 (2012).

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

[15] Brodahl, C., Hadjerrouit, S. and 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).

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

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

[18] 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).

推薦文

本論文は，初学者が容易に使えるように設計されたグループプログラミングを支援するシステムの提案と評価を行っている．このシステムではグループの各メンバが独立したブランチを持ち，各自のタイミングで他者のコードを参照・コピーできる「独立同期モデル」を用いることにより，初学者が容易に利用できる協調プログラミング環境を提供している．これにより，グループ演習で起こりがちな，特定のメンバのみがコードを書き，その他はほとんど参加していないという問題を解消し，能力にかかわらず個々のメンバが遠慮せずに満足度の高い貢献ができたという評価が得られている．本論文で扱っているようなグループでのプログラミング演習を行い，その運営に問題意識を持っている教員は多い．本論文の成果はそれを解消していくための大きな参考になるものであり，また，プログラミング以外の協調作業に対しての適用も考えることができるものであるので，非常に意義のあるものとなっている．

（論文誌「教育とコンピュータ」編集副委員長/
コンピュータと教育研究会主査　西田知博）

加藤 優哉　（学生会員）

2014 年静岡大学情報学部卒業．2016 年静岡大学大学院情報学研究科修士課程修了．現在，（株）東芝に在職中．在学中はプログラミング学習および協調学習の研究に従事．

松澤 芳昭　（正会員）

2002 年慶應義塾大学大学院政策・メディア研究科修士課程修了．2007 年同後期博士課程単位取得退学，博士（政策・メディア）．2008 年より静岡大学情報学部特任助教，同学部助教，講師を経て，2015 年より青山学院大学社会情報学部助教．情報システム学を応用したユーザ中心のソフトウエア設計と開発，情報教育，学習科学の研究に従事．日本教育工学会，情報システム学会，教育システム情報学会各会員．

酒井 三四郎　（正会員）

1984 年静岡大学大学院電子科学研究科博士後期課程修了．学習院大学，新潟産業大学，静岡大学工学部を経て，1998 年静岡大学情報学部助教授．現在，同学部教授．工学博士．ソフトウェア開発支援環境，プログラミング学習支援環境，遠隔学習，協調学習に関する研究・開発に従事．電子情報通信学会，教育システム情報学会各会員．