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

加藤 優哉[1,a)]　松澤 芳昭[2,b)]　酒井 三四郎[2,c)]

概要：初学者の協調プログラミングを支援するシステムの設計を行った．本システムが支援する協調プログラミングは，(1) 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) 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．本研究が提案する「独立同期モデル」の特徴は，(A) 自分の変更が相手に直接影響を与えない，(B) 常にメンバの最新のプログラムを閲覧可能である，(C) プログラムを単純な方法で取り込むことが可能である，という 3 点である．本モデルに基づくシステムを開発し，プログラミング入門講義を受講する文科系の学生 6 名（3 名 2 グループ）を対象に使用実験を実施した．使用過程の記録，およびインタビューの質的分析より，本システムが初学者に利用可能な使用容易性をもち，限定的ではあるが協調プログラミングを支援することができたことが確認できた．

キーワード：協調学習，プログラミング教育，初学者

Promoting collaborative programming for novices by independent and synchronization approach

Abstract: We have developed a system to support collaborative programming for novice. We defined collaborative programming as (1) all members make any contribution to the group products on an assumption of skill level differences between members, and (2) a product is not a result of the work by completely isolated work sharing, but the procedure includes activities of reading others' code, improvement, and expansion. We proposed the independent and synchronization model to support such a collaborative programming. The proposed model has following features: (A) changes made by someone do not directly affect others' code, (B) it is always available to view others' latest programs, (C) novice programmers can import others' program in a simple way. We implemented the model as a workable software, and it was examined for 6 students (3 members per group) of liberal arts in our introductory programming course. Qualitative analysis was conducted by using system logs and a result of interviews for users. The result shows that the system usability is acceptable level for novice programmer. We also found that interaction logs between students showed that the system succeeded to promote a collaborative programming though limited situations.

Keywords: Collaborative Learning, Programming education, novice

1. はじめに

プログラミング入門教育の場では，総合的な応用力の育成を目的として，グループを組んでプログラミングを行うことがある [1][2]．本論文ではこのようなプログラミングを協調プログラミング（2 章で詳説）と呼ぶ．J. Morrison らによれば，グループでプログラミングを行うことによって以下の機会を得ると言われている [3]．1）グループのコミュニケーションとマネジメントスキル，2）モチベーションの向上，3）協調学習，4）知識のギャップを埋める，5）より現実的なプロジェクト，6）反対主張の経験，7）グループプログラミング環境での技術的な問題の理解，8）コード管理プロセスの理解．

1 静岡大学大学院情報学研究科
Graduate School of Informatics, Shizuoka University
2 静岡大学情報学部
Faculty of Informatics, Shizuoka University
a) kato@sakailab.info
b) matsuzawa@inf.shizuoka.ac.jp
c) sakai@inf.shizuoka.ac.jp

しかしながら，著者自身の経験と講義の観察から，A）グループ内で最もプログラミングが得意な学習者（1 番手）がプログラムを全て組み，他のメンバ（2 番手，3 番手）はプログラムを組まない，B）グループメンバ全員が完全に独立して作業を行う，という現象がある．

原因として，初学者が既存のツールを用いたファイルの共有を煩わしく感じることが考えられる．グループでのプログラミングを支援する構成管理ツールも存在するが，初学者はその存在を知らない．存在を知っていたとしても，既存の構成管理ツールは初学者にとって習得難易度が高い．

協調プログラミングを支援するにあたり，その実態を解明する必要がある．しかしながら，協調プログラミングの実態は解明されていないことも問題である．

従って，本研究の目的は，協調プログラミングの促進と実態解明である．

2. 協調プログラミング

2.1 定義

協調プログラミングについて，Cockburn らは，2 人のプログラマが 1 つのコンピュータを用いて side-by-side でソフトウェア開発を行うことと定義している [4]．この定義は，ペアプログラミングの手法と重複する部分がある．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 [5]．

本研究において協調プログラミングは，以下のように定義する．

(1) メンバの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) 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．

2.2 環境

我々が行っているプログラミング入門講義の協調プログラミングについて説明する．我々は，講義の最終プロジェクトとして協調プログラミングを実施している．受講者は，本講義の最終プロジェクトで協調プログラミングを始めて経験する文科系の学生である．1 グループあたりの人数の上限を 3 名として，学生に任意にグループを組ませている．

2.3 現状調査

最終プロジェクト終了後，受講者に協調プログラミングに関するアンケートを実施した．有効な回答は，グループを組まずに個人で最終プロジェクトに取り組んだ 1 人を除いた 70 件とした．

表 1 協調プログラミングで苦労したこと

ソースコードの結合	36 人（51%）
役割分担	34 人（49%）
メンバのソースコードの解読	26 人（37%）
メンバとのコミュニケーション	21 人（30%）
ファイルの共有	15 人（21%）
その他	4 人（6%）

表 2 ファイル共有の方法

USB メモリのような外部メモリ	42 人（60%）
Skype のようなメッセンジャー	28 人（40%）
メール	21 人（30%）
Dropbox のようなクラウドサービス	5 人（7%）
その他	8 人（11%）

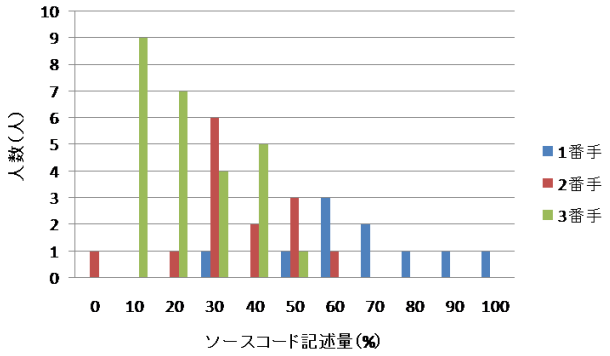

図 1 能力とソースコードの記述割合の関係

2.3.1 苦労したこと

協調プログラミングを経験して苦労したことに関する質問の回答結果を**表 1** に示す．約半数の学生がソースコードの結合や役割分担に苦労を感じていると答えている．

ソースコードを結合する際に，ファイルを共有する必要がある．ファイルを共有することに煩わしさを感じ，その結果としてソースコードの結合に苦労を感じたと回答した学生が存在することが考えられる．

2.3.2 ファイル共有の方法

協調プログラミングを行う際のファイル共有の方法に関する質問の回答結果を**表 2** に示す．60%の学生が外部メモリによってソースコードのやり取りを行っていると答えている．その他の回答は，1 台のコンピュータを共有して作業を行ったというものだった．

外部メモリによってファイルを共有する場合，直接会って手渡しする必要がある．従って，60%の学生が任意のタイミングでファイルを共有することができず，お互いに都合の合う時間を調整して外部メモリを渡していたと考えられる．

2.3.3 能力差から生じるソースコード記述割合の差

グループ内のプログラミング能力の位置づけと，グループのソースコードの記述量の関係を表したグラフを**図 1** に示す．プログラミング能力とソースコード記述量につい

て，回答者自身の自己分析で答えるアンケートから得られた結果である．アンケート結果から 3 名グループの回答を抽出して得られたグラフである．

グループ内のプログラミング能力の位置づけについて，1 番手という回答が 10 件，2 番手という回答が 14 件，3 番手という回答が 26 件であった．ソースコードの記述割合の平均値は，1 番手が 67%，2 番手が 35%，3 番手が 23%であった．2 番手，3 番手が，1 番手と比べて半分程度のソースコード記述量であることが分かる．

2.4 問題分析

2 番手や 3 番手にあたるメンバがグループに実力に見合った貢献ができない理由として，1 番手にあたるメンバに遠慮をしているという解釈ができる．2 番手，3 番手は個々にプログラムに関する意見を有しているにも関わらず，ソースコードを編集しづらいということは，フラストレーションの蓄積にも繋がる．

以上の問題を解決するためのアプローチとして，前述したソースコードの結合やファイル共有に関する苦労を解消するツールの提供が考えられる．ソースコードの結合について，初学者に可能な単純操作でソースコードの結合が可能となる仕組みを提供する必要がある．ファイルの共有について，外部ツールを利用することなくファイルの共有が可能となる仕組みを提供する必要がある．

我々は，問題を解決することによって，2 番手，3 番手がグループに貢献し，作品の完成度がより高いものになるという仮説を立てた．

3. 先行研究

協調プログラミング支援という観点から先行研究を述べる．

黎らは，グループプログラミングでの教育を支援するために，Web 上で電子教科書，仕様書，進捗状況，掲示板などを管理するシステムを開発している [6]．このシステムで，システム設計の方法や仕様書の書き方などを身につけることができると主張している．

黎らの研究では，グループで開発を行う際の上流行程（仕様書や設計）について身に付き，メンバ間のコミュニケーションの支援も可能なシステムを実装している．しかし，このシステムではコーディング面は考慮されていない．

Eclipse 上でソースコードの同期編集を可能とするプラグイン Saros を S. Salinger らが開発している [7]．Saros は 1 つのプロジェクトをグループのメンバで共有し，全員で 1 つのファイルを編集することが可能である．指定したメンバの視野（スクロールやファイルの移動）を追従する機能も実装されている．以上の機能より，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．

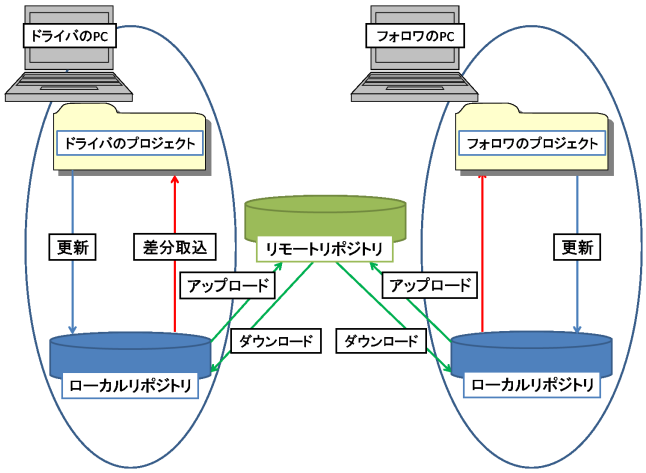

図 2 コミットモデル

S. Salinger らの研究は，グループメンバの意識共有という点で有用であり，遠隔でのペアプログラミングに加え，グループでのソースコードをレビューし合うようなシーンでも活用可能なシステムである．しかし，1 つのプロジェクトをグループで共有することから，自身の変更が直接グループのソースコードに反映されるため，気軽にソースコードを編集することができない．

青木らは，分散型ペアプログラミング演習システムを開発している [8]．ペアプログラミング形式に習って，ドライバとナビゲータの役割が割り振られ，ドライバのみがソースコードの編集が可能である．アノテーションやチャット機能というコミュニケーションを支援する機能が実装されている．

同様な分散ペアプログラミングを支援するシステムとして Sangam[9] や XPairtise[10] が開発されている．これらのシステムは，Saros と同様に 1 つのプロジェクトをペアで共有するモデルが採用されている．従って，2 番手以下のメンバが気軽にコーディングに参加できないという Saros 同様の問題点が存在する．

以上のとおり，グループでのプログラミングにおいて，コミュニケーションを支援するようなシステムや分散ペアプログラミング支援するようなシステムは開発されている．git などのプロフェッショナル向けの構成管理ツールも存在する．しかしながら，初学者に対応し，本稿で定義する「協調プログラミング」を支援するようなシステムは開発されていない．そこで本研究では，初学者が利用可能な協調プログラミング支援システムの設計と開発を行う．

4. 独立同期モデルの提案

本研究では，初学者の協調プログラミングを支援するための独立同期モデルを提案する．既存システムのモデルと独立同期モデルの比較を行い，特徴・問題点を述べる．

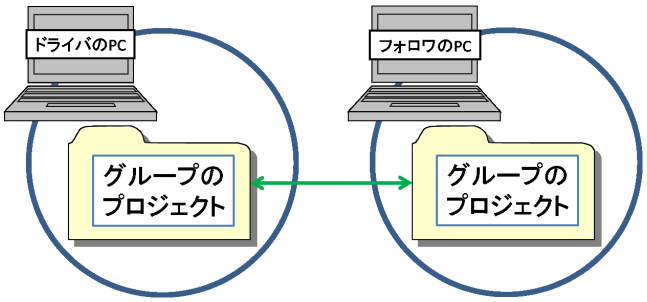

図 3　共同編集モデル

4.1　コミットモデル

既存の構成管理ツールの例として git と Saros を挙げる．git のようなモデル（以下，コミットモデル）を**図 2** に示す．コミットモデルの特徴は次の 4 点である．

(1) ローカルの変更をリモートに反映させるために push を行う必要がある．
(2) リモートの内容をローカルに反映させるために fetch を行う必要がある．
(3) merge で競合が発生したら解決する必要がある．
(4) 1 人が複数のバージョンを管理することが可能である．

コミットモデルの問題点の 1 つとして，特徴に示したような push，fetch，merge などの様々な用語の概念や操作方法を学ばなければ使いこなせない点が挙げられる．特徴 (1)，(2) について，グループメンバの最新版を見るためには，最新版が commit，push されていることが前提となる．メンバに最新版を push するように要求することは，そのメンバの作業を一時的に阻害してしまうことが問題である．このようなことを気にして push するように要求が出せないことが起こることも考えられる．

4.2　共同編集モデル

Saros のようなモデル（以下，共同編集モデル）を**図 3** に示す．共同編集モデルの特徴は次の 4 点である．

(1) グループ全員で 1 つのプロジェクトを編集する．
(2) 同じファイルに複数人が同時に書き込み可能である．
(3) 編集の様子がリアルタイムに表示される．
(4) メンバの視野（スクロールやファイルの切り替えなど）を共有できる．

共同編集モデルの問題点は，個人の編集がグループのファイルに直接変更を加えるため，2 番手や 3 番手にあたるようなメンバがソースコードに手が出しづらく，気軽にプログラムを組めない点である．

4.3　独立同期モデル

独立同期モデルを**図 4** に示す．独立同期モデルは，次の特徴をもつ．

(1) 自分の変更が相手に直接影響を与えない．
(2) 常にメンバの最新のプログラムを閲覧可能である．

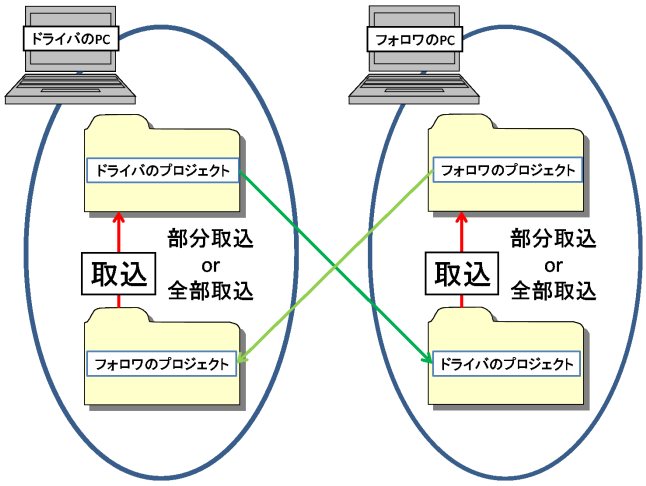

図 4　独立同期モデル

(3) プログラムを単純な方法で取り込むことが可能である．

特徴 (1) より，自分自身が所有するファイルは，他のグループメンバのファイルとは独立しているため，共同編集モデルの問題点であるソースコードに手が出しづらいという問題点が解消される．特徴 (2) より，グループメンバに最新版の反映の要求をすることなく，常に最新版の閲覧・取り込みが可能である．従って，要求によって作業を一旦停止させられることなく，相手に気を遣いながら要求をすることもなくプログラミングに取り組むことができる．特徴 (3) より，取り込み操作の複雑性が解消される．以上より，コミットモデルの問題点が解消可能であると考えられる．

git の merge のような 2 つのソースコードの差分の取り込み（差分取込）については，ユーザが，グループメンバのソースコードの任意の箇所をコピーし，自分のソースコードの任意の箇所にペーストするという部分取込で対応する．

以上より，独立同期モデルは初学者の協調プログラミングを支援できるものと考えられる．

5.　提案システム：CheCoPro

5.1　設計目標

本研究の目的は，初学者の協調プログラミングを支援するシステムの開発である．提案するシステム CheCoPro (Cheeful Collaborative Programming) は独立同期モデルを採用し，既存の構成管理ツールとは異なる，単純な操作と概念で利用可能であることが設計目標である．

5.2　実装

本システムは，静岡大学のプログラミング入門講義で用いている開発環境「論プロエディタ」の拡張として実装した．CheCoPro のインタフェースを**図 5** に示す．主な機能は同期機能，グループメンバ観察機能，取り込み機能である．

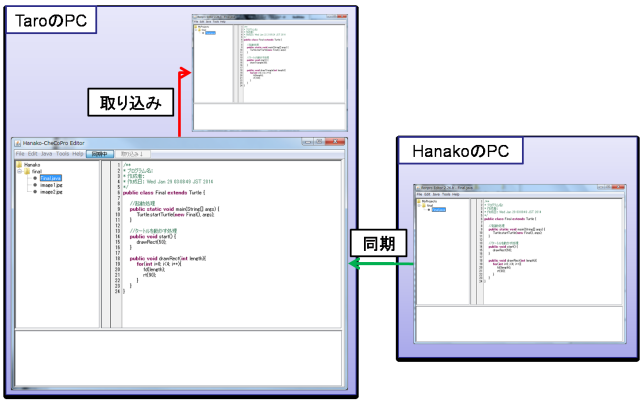

図 5　インタフェース

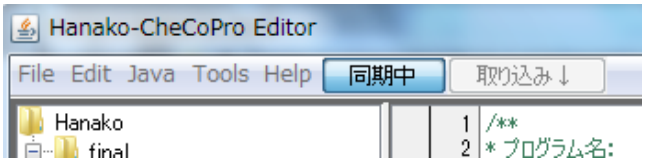

図 6　同期中

図 7　非同期中

5.3　同期機能

協調プログラミングの定義（1）メンバの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という定義に対して，同期機能を実装した．自分以外のメンバのファイルは同期されている状態であり，ファイルを編集することで直接他のメンバに影響を与えることはない．従って，個人のペースで自由なコーディングが可能である．同期中と非同期中の論プロエディタのメニューバーを**図 6** と**図 7** に示す．最新版のものを受け取るか否かはボタンで切り替えることが可能である．

5.4　グループメンバ観察機能

協調プログラミングの定義（2）完全な分業ではなく，他の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という定義に対し，グループメンバのソースコード編集の様子のリアルタイム表示機能を実装した．グループメンバのソースコードを観察する様子を**図 8** に示す．グループメンバがソースコードを編集する様子を，自分自身が編集を行っているウィンドウ（論プロエディタ）とは別のウィンドウ（CheCoPro エディタ）で閲覧が可能である．

5.5　取り込み機能

協調プログラミングの定義（2）を支援するため，部分取込と全部取込という 2 種類の単純な取り込み機能も実装した．部分取込は，CheCoPro エディタから取り込みたい部分をコピーし，論プロエディタ上にペーストすることで実現可能である．全部取込は非同期中に図 7 右の取り込みボタンを押すことで可能である．取り込みボタンを押すと**図 9** に示す取り込みダイアログが開かれ，ソースコードを取り込むか素材を取り込むかが選択可能である．全部取込は，同名ファイルは上書きされる仕様となっている．

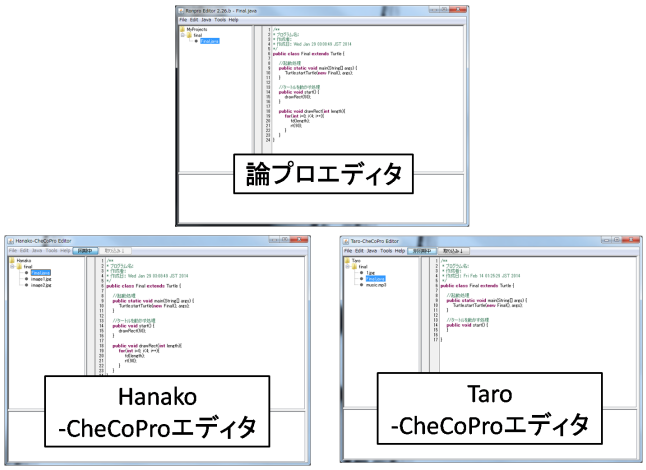

図 8　グループメンバのソースコードの閲覧

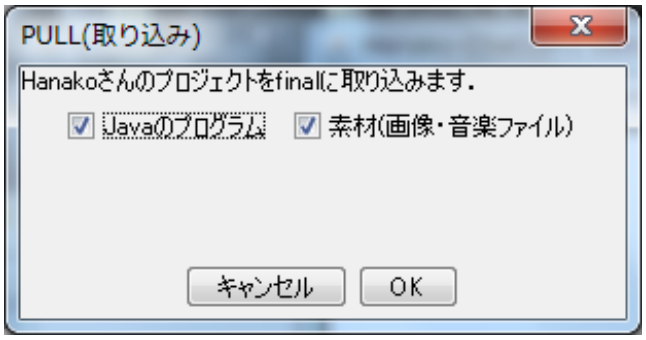

図 9　取り込みダイアログ

6.　実験方法

本学のプログラミング入門講義の場で提案システムの評価実験を行った．本章ではその方法を述べる．

6.1　対象

プログラミング入門講義を受講し，最終課題にエントリした全 39 グループの文科系大学生から 6 名（3 名 2 グループ）を対象とした．被験者は Java の学習を始めて 4 ヶ月であり，オブジェクト指向の概念を除いた Java プログラミングの基礎について学習済みである．協調プログラミングを行うことは始めてである．被験者を抽出する際に，学力等は考慮せず無作為に抽出した．

表 3　プロジェクトの概要

	成功グループ	失敗グループ
ソースコードの行数	364	2510
自作メソッド数	3	31
素材ファイル数	20	239
作品種別	相性診断	格闘ゲーム
構成人数	3	3

図 10　成功グループの作品

6.2　実験内容

プログラミング入門講義の最終課題に取り組む際に提案システムの評価実験を実施した．最終課題の内容は，Java を使った GUI 作品の制作である．システムの初期設定と説明は 20 分程度で行った．実験期間は 1 週間であり，その様子の観察を行った．制作物の完成後にグループでのプログラミングに関するアンケートを実施した．

6.3　利用したデータ

協調プログラミングの定義を満たしてプログラミングを行うことができていたグループを成功グループ，満たしていなかったグループを失敗グループとして，利用したデータについて説明する．

各グループのプロジェクトの概要を**表 3** に示す．ソースコードの行数，自作メソッド数，素材ファイル数の全てにおいて，失敗グループが成功グループを上回るものが制作された．

成功グループが制作した作品の 1 画面を**図 10** に示す．成功グループの作品は，質問と選択肢を表示して選んだ選択肢に応じて次の質問や結果を表示するものである．

失敗グループが制作した作品の 1 画面を**図 11** に示す．失敗グループの作品は，格闘ゲームである．パンチやキックなどのコマンドをキーボードに割り当て，2 人プレイが可能である．

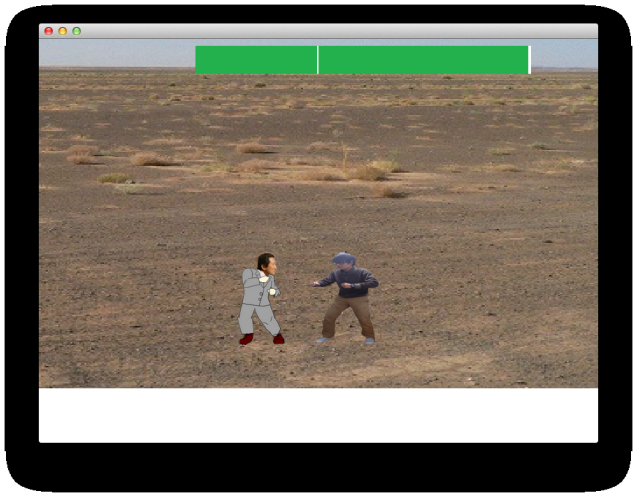

図 11　失敗グループの作品

表 4　ファイル共有の方法（CheCoPro 利用群）

CheCoPro	4 人（67%）
Skype のようなメッセンジャー	2 人（33%）
メール	0 人（0%）
Dropbox のようなクラウドサービス	0 人（0%）
USB メモリのような外部メモリ	0 人（0%）
その他	0 人（0%）

表 5　協調プログラミングで苦労したこと（CheCoPro 利用群）

役割分担	4 人（67%）
メンバのソースコードの解読	3 人（50%）
メンバとのコミュニケーション	2 人（33%）
ファイルの共有	1 人（17%）
ソースコードの結合	1 人（17%）
その他	0 人（0%）

7.　実験結果

7.1　アンケート結果

協調プログラミングに関するアンケート結果のうち，ファイル共有の方法を**表 4** に示す．被験者は CheCoPro を利用していると答えている．

協調プログラミングで苦労したことに関する質問の回答結果を**表 5** に示す．表 5 より，ファイルの共有やソースコードの結合を解消できたと判断している．

提案システムに関するアンケートについて，他ツールを使用せずにグループメンバのファイルを受け取ることができる点を 6 名中 5 名が利点として答えた．自由記述欄には，グループメンバと事前に打ち合わせをするか連絡を取り合うようなことをしないと，他のメンバのソースコードを解読することが難しいということから，チャット機能を要望する意見があった．

7.2　ソースコードのやり取り

取得したログとインタビューから得られた取り込みの流

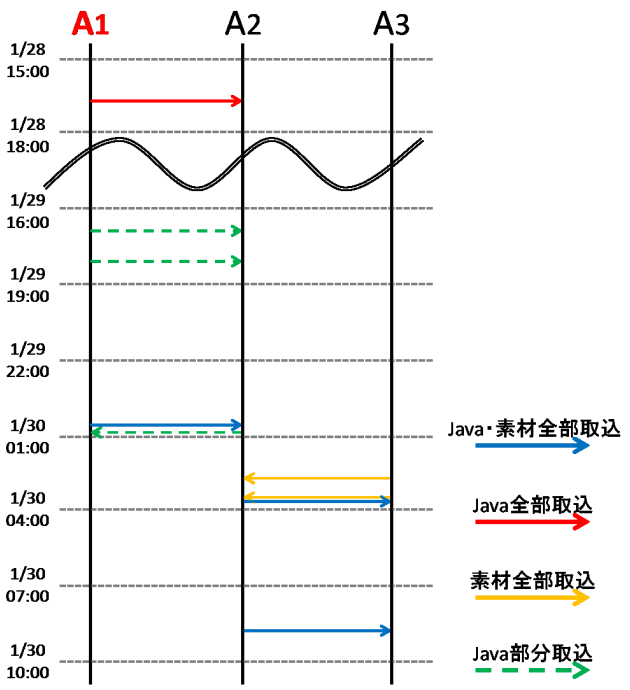

図 12　成功グループの取り込み流れ図

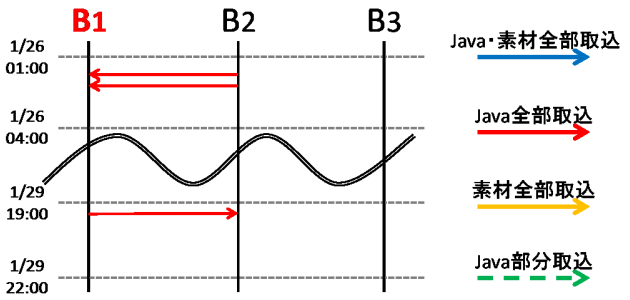

図 13　失敗グループの取り込み流れ図

れ図を，成功グループについて**図 12** に，失敗グループについて**図 13** に示す．各グループのリーダーは A_1 と B_1 である．どちらのグループも，システムの使い方が理解できないということはなく，成功グループに関しては積極的に取り込みが行われている．

7.2.1　成功グループのやりとり

成功グループに関して，A_1 が主体となってプログラムを組む様子が観察できた．A_2 は A_1 のソースコードを取り込み，取り込んだソースコードを基に変更を加えて実行するという操作を繰り返していた．成功グループは相性診断のようなプログラムを制作した．A_2 は渡された引数によって診断結果を表示するメソッドを制作していた．A_1 がメイン動作を制作していたため，A_2 はメイン動作部分を取り込んで制作したメソッドの動作を試すような利用方法も観察できた．A_3 はコーディング面では参加せず，素材を提供し，最終的に完成したプロジェクトを取り込み，動作のテストを行っていた．

7.2.2　失敗グループのやりとり

失敗グループに関して，B_1 が主体となってプログラムを組む様子が観察できた．失敗グループは CheCoPro だけではなく，メッセンジャーを利用してファイルを交換していた．B_2 のソースコードを B_1 は CheCoPro で 2 回取り込んでいる．このとき，B_1 と B_2 は同じ内容のプロジェクトを持っていた．B_1 は新しいクラスを作成し，そこに B_2 のソースコードをコピーすることで，同じ内容のクラスを 2 つ所持した．予備のファイルを持つような利用方法が観察できた．B_3 はプロジェクトに参加していないが，インタビューより体調を崩していたことがわかった．

8.　考察

8.1　独立同期モデルの有効性

8.1.1　協調プログラミングの定義（1）の達成度

成功グループ，失敗グループ共にメンバの能力は一律ではなかった．成功グループについては，成果物に個々の実力に見合った貢献が見られた．失敗グループに関して，成果物に B_2 と B_3 の貢献は見られなかった．B_2 は最終的に貢献はできなかったが，リーダーが記述したソースコードを自身で試行錯誤し，試験的に書き換えている様子が観察できた．B_3 は体調不良という外部的な要因でプログラムに手をつけることができなかった．以上より，独立同期モデルは協調プログラミングの定義（1）をおおむね支援可能であると考えられる．

8.1.2　協調プログラミングの定義（2）の達成度

成功グループに関して，積極的にソースコードや画像等の素材を取り込んでいる様子が観察できた．A_2 に関してはプログラムを拡張させる様子も観察できた．失敗グループについては，メンバのソースコードを観察する様子は見られたが，それを取り込んで拡張するという様子は見られなかった．従って，定義（2）についても限定的ではあるが支援可能であると考えられる．

8.2　CheCoPro の使用容易性

20 分程度の導入・説明で CheCoPro の利用方法を理解し，ソースコードや素材ファイルの取り込みが行われている．このことから，CheCoPro は初学者に利用可能な使用容易性があることがわかった．

8.3　CheCoPro の限界

成功グループは失敗グループに比べて，コード量が少なく，素材ファイルも少ない．制作した作品について，質問と選択肢を表示して選んだ選択肢に応じて次の質問や結果を表示する単純な条件分岐の作品となっている．グループリーダーと他のメンバとの能力差が，失敗グループと比べて小さかったために 2 番手，3 番手メンバがソースコードを読むことができ，積極的に協調プログラミングに参加す

```
ImageTurtle wn[] = new ImageTurtle[6];
ImageTurtle wf[] = new ImageTurtle[5];
ImageTurtle wb[] = new ImageTurtle[5];
ImageTurtle sq = new ImageTurtle("s.png");
ImageTurtle wp[] = new ImageTurtle[2];
ImageTurtle wps[] = new ImageTurtle[2];
ImageTurtle wpd[] = new ImageTurtle[2];
ImageTurtle wk[] = new ImageTurtle[2];
ImageTurtle wkd[] = new ImageTurtle[2];
```

図 14 失敗グループの変数宣言の一部

```
for(int i = 0; i < 2; i++){
 wkd[i] = new ImageTurtle("kd00" + i + ".png");
}
for(int i = 0; i < 4; i++){
 wku[i] = new ImageTurtle("ku00" + i + ".png");
}
for(int i = 0; i < 2; i++){
 wsqp[i] = new ImageTurtle("sp" + i + ".png");
}
```

図 15 配列への素材ファイルの格納

ることが可能であったと考えられる．

失敗グループに関して，グループリーダーの能力が他のメンバよりも高く，制作した作品が成功グループに比べて複雑であったた．コード量は 2510 行であるが，変数や一部メソッドの先頭に一言コメントが書かれている程度であった．失敗グループの作品が格闘ゲームということで，パンチやキックなどのアクションを行うことでキャラクタの画像が切り替わるような仕様である．従って，素材数 239 個という量になっている．素材のファイル名について，“d001.png”や“ku003.png”のように，他のメンバが見てどのような画像か分かるような名前付けがされていなかった．失敗グループのリーダーが記述した配列の定義を**図 14** に，配列に素材ファイルを格納するために記述したコードを**図 15** に示す．各アクション毎に配列を用意し，そのアクションに関連する素材ファイルを図 15 のように格納している．変数名でどのようなアクションか読み取ることができない上，コメントも記述されていない．

他のメンバがソースコードを解読できるようなプログラムが組まれていなかったことが，リーダーが単独でコーディングを行うことになった原因であると考えられる．

8.4 今後の課題

独立同期モデルは，限定的ではあるが協調プログラミングの支援を行うことが可能であることがわかった．

今回の実験では被験者は 6 名であった．実験により抽出された問題点の信頼性の向上と協調プログラミングの更なる実態解明を行う上で，取り込みログのビジュアライズ機能を実装し，被験者を増やして再び実験を行う必要がある．

9. 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．現状では，協調プログラミングを行うことが初学者にとって困難となっている．そこで本研究では独立同期モデルを提案し，初学者にも扱うことが可能な協調プログラミング支援システム CheCoPro を開発した．

独立同期モデルの有効性を検証するために，プログラミング入門講義にて使用実験を行った．その結果，初学者にとって利用可能な使用容易性をもち，限定的ではあるが協調プログラミングの支援が可能であることがわかった．システムのログから，協調プログラミングの実態調査が可能であることがわかった．

今後，被験者を増やし，協調プログラミングの実態調査を進めると共に，システムに必要な機能を抽出して実装することを目指している．

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] J. Morrison: *FACILITATING COLLABORATIVE LEARNING WITHIN PROGRAMMING PROJECTS*, Issue in Information System, Volume V, No 1, pp.221-225 (2004).

[4] A. Cockburn, L. Williams: *The Costs and Benefits of Pair Programming*, in First International Conference on Extreme Programming and Flexible Processes in SoftwareEngineering (XP2000) (2001).

[5] 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).

[6] 黎翔寧，宮寺庸造，横山節雄，櫨山淳雄：グループプログラミング教育支援システムの開発，電子情報通信学会技術研究報告，ET，教育工学，Vol.98，pp.81-88 (1998).

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

[8] 青木一浩，立岩佑一郎，山本大介，高橋直久：分散型ペアプログラミング演習システムの実現，電子情報通信学会技術研究報告，ET，教育工学，Vol.109，pp.193-198 (2009).

[9] C. W. Ho, S. Raha, E. Gehringer, L. Williams: *Sangam: a distributed pair programming plug-in for Eclipse*, In eclipse ’04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange, pp.73-77 (2004).

[10] T. Schümmer, S. Lukosch: *Supporting the social practices of distributed pair programming*, In Groupware: Design, Implementation, and Use, volume 5411 of Lecture Notes in Computer Science, pp.83-98 (2008)