Vol.2013-CE-118 No.9
2013/2/8

アジャイル型ソフトウェア開発 PBL における CCBR を拡張したコードレビュー支援環境の提案

木崎悟[†1]　田原康之[†1]　大須賀昭彦[†1]

本研究では，アジャイル型ソフトウェア開発 PBL における学習者のコードレビュー支援を目的とする．実社会のコードレビューは，レビュアーは経験豊富なエンジニアが担当し，経験の少ないエンジニアを指導することが適切である．コードレビューにより，ソフトウェア品質を高めると同時に開発スキルの向上を図ることができる．PBL では学生同士がレビューすることが前提となる．しかし，実務経験がない学生の場合，レビュアーの選定が難しい．また，何を基準とするのか，どこからレビューするかわからないといった問題が起こり，コードレビューを実施しても指摘できる箇所も不十分となってしまう．このため本論文では，ソフトウェア開発における既存のレビュー手法（CCBR）をソフトウェア開発 PBL 向けに拡張し，プロジェクト成果物のソフトウェア品質と学習者の教育効果を高めるコードレビューのモデルを提案する．

The approach of code review support environment which extended CCBR in an agile software development PBL

SATORU KIZAKI[†1]　YASUYUKI TAHARA[†1]　AKIHIKO OHSUGA[†1]

In this paper, we aim to support a student's code review support in an agile software development PBL. An engineer with abundant experience takes charge of reviewer, and the code review of an actual world guides an engineer with little experience. Therefore, we can raise software quality, and it can be improvement in development skill. In PBL, students review each other. However, selection of reviewer is difficult when it is a student without an experience in actual business. Moreover, they become insufficient which can be pointed out even if it enforces a code review. For this reason, the existing review technique (CCBR) in software development is extended for a software development PBL, and the model of a code review which heightens a software quality of a project product and a student's education effect is proposed.

1. はじめに

近年，実社会で即戦力として活躍できる人材を育成することを目的に Project Based Learning（以下，PBL とする）が実施されている．PBL では明確な目標を掲げて，実際の業務の内容に近い形で 1 つのプロジェクトを完成させていくプロセスの中で，社会で役立つスキルやノウハウを習得する．情報系教育機関においても，PBL 形式で授業が実施される例も増えている[1, 2, 3].

慶應義塾大学 SFC では，学部生を対象に PBL 型の科目「協創型ソフトウェア開発」を実施している．この授業は，2005 年度秋学期に開講され，2012 年度で 7 回目である．毎年，複数の開発チームが結成され，プロジェクトのメンバは 3〜5 名で構成される．また，プロジェクトマネージャは社会人技術者が担当し，実際にシステムを必要とする企業や個人の顧客が設定され，ユーザに利用されるソフトウェアの開発を目標としている[4, 5].

また，2010 年度までは，ウォーターフォール型などの非アジャイル型開発プロセスを利用していたが，2011 年度以降は，実社会の変化に対応するために，アジャイル型開発手法を採用した[6]．本論文では，アジャイル型ソフトウェア開発 PBL における成果物に対する品質と学習者の教育効果を高めるコードレビューのモデルを提案する．

2. アジャイル型ソフトウェア開発 PBL

2.1 アジャイル開発

2010 年の米国フォレスター・リサーチ社[a]の調査[7]によると，米国におけるアジャイル開発の採用は 35%となっており，ウォーターフォール型の採用率 13%を大きく上回っている．1990 年代まではウォーターフォール型が主要な開発手法であった．しかし，完成したソフトウェアのほとんどの機能が利用されていない．

また，開発コストと保守運用コストが要求変更に使われていることから，顧客に必要な機能を提供することと，要求変更に柔軟に対応する目的で，アジャイル開発手法が誕生した．アジャイル開発手法には，エクストリーム・プログラミング（以下，XP）[8]やスクラム[9]などが考案されている．

2.2 スクラム

スクラムとは，欧米などで最も多く普及しているアジャイル開発手法である．竹内弘高氏・野中郁次郎氏による日本製品開発における研究[10]が基礎となり，J.Sutherland により，ソフトウェア開発プロセスとして体系化した．その名の通りチームが一丸となって開発を進められるように，プロジェクト運営の側面に焦点を当てている．

メンバの役割は，責任者として要件と優先度を決める「プ

†1 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

a) フォレスター・リサーチ（Forrester Research）とは，技術や市場調査を得意とする米国の独立系のアナリスト・ファーム．

ⓒ 2013 Information Processing Society of Japan

ロダクトオーナー」,プロジェクトが円滑に進むようにサポート（アドバイス）をする「スクラムマスター」，開発者の集団である「チーム」から構成されている．

進め方は，プロダクトオーナーが要件を「プロダクトバックログ」と呼ばれる機能単位に分割する．機能に関してはストーリー形式で記述する．そして，個々の機能に対して優先度と完了の定義を決める．プロダクトバックログには，製品の開発，リリースのために必要なすべての事柄を含める．次に，スクラムマスターとチームはプロダクトバックログの機能を実現するために実施する作業項目（タスク）を洗い出して,「スプリントバックログ」を作成する．スプリントバックログは，プロダクトオーナーが決めた機能をより詳細化する．内容はスプリント計画会議で決定される．スプリントとは，最大 4 週間と定められた開発期間のことを指す．各スプリントの長さは一定であり，1 つのスプリントで出荷可能な製品を作成する．

また，毎日の作業は，デイリースクラムから始まる．デイリースクラムでは，作業報告と問題共有をチームのメンバ全員で行う．そして，タイムボックスという一定の時間を決めて作業する．作業終了時には，KPT 法[11]などにより，レトロスペクティブ（振り返り）を行う．

2.3 プラクティスの PBL 適用時の問題

アジャイル開発では，プラクティスを取り入れることで適用する効果がでるとされている．プラクティスとは，経験に基づいて効果が認められた開発のスタイルのことである．2.2 で紹介したスクラムでは，定められたプラクティスはないが,以下は有効であると評価されている.しかし，PBL に適用する場合は問題が発生する．

- テスト駆動開発（TDD: Test-Driven Development）

テスト駆動開発とは，最初にテストを書き，そのテストが動作する必要最小限な実装をとりあえず行った後，コードを洗練させていくプラクティスである[12].

しかし，既存システムに対する機能追加などは，テスト駆動開発を導入することは難しい．また，プログラミングに慣れていない学生の場合は，何をテストすればいいかわからないといった問題がある．

- ペアプログラミング（Pair Programming）

ペアプログラミングとは，開発者が 2 人でレビューをしながら，コーディング作業を行う手法である．キーボードを操作してコードを書く人を「ドライバー」といい，ドライバーに対してアドバイスを出す人を「ナビゲーター」と呼ぶ．プログラミング教育の観点から見ると，ペアプログラミングは有効な手段であると考えられている．ペアプログラミングでは,ナビゲーターがレビュアーになることで,ドライバーに対する教育効果が期待できる．

また，分散開発環境の場合は，共同で開発することができない．既存研究では，リモートペアプログラミングによる研究がされている[13].

2011 年度のプロジェクトにおいてもペアプログラミングを取り入れたチームがあり，実施した学生より『プログラミング経験がないため，簡単なコーディングでもつまずいてばかりで苦しくなることがあった．しかし，メンバとペアプログラミングをすることで，問題点がすぐに見つかり，コーディングが苦にならなくなった』と報告されている．

しかし，両者が経験不足の場合は，ナビゲーターも的確なアドバイスができず，教育効果が期待できない．また，授業内での作業は時間が限られてしまう．

- 継続的インテグレーション（Continuous Integration）

継続的インテグレーション（以下，CI）とは，1 日に何度もビルドを実行して，発生する問題を早期に検出し，フィードバックサイクルを短くして，ソフトウェア開発の品質と生産性を向上させる仕組みである．CI を実現するツールは，Jenkins[14]が有名である．しかし，初期の導入コストがかかるという問題もある．

- リファクタリング（Refactoring）

リファクタリングとは，プログラムの動作を変えずに，ソースコードの内部構造を整理することである．リファクタリングを行うタイミングであるが，定期的なコードレビューにより,他者の指摘などを踏まえてコードを修正する．

しかし，後述の理由により，アジャイルではコードレビューを定期的に実施することが難しい．

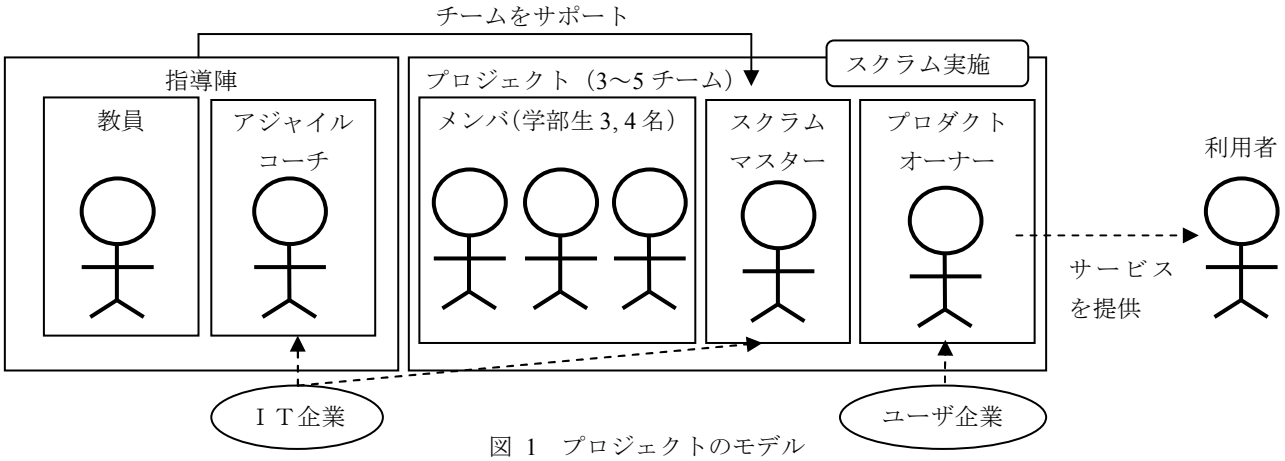

図 1　プロジェクトのモデル

Figure 1　Model of a project in the proposed PBL system.

ⓒ 2013 Information Processing Society of Japan

2.4 協創型ソフトウェア開発

アジャイル型ソフトウェア開発 PBL「協創型ソフトウェア開発」について述べる．本授業はアジャイル型開発手法「スクラム」を採用している．

本授業のプロジェクトモデル（図 1）では，プロジェクトでは，顧客（プロダクトオーナー）を設定している．また，IT 企業からスクラムマスターが参加して各チーム（3～5）に加わる．そして，教員の他にアジャイルの専門家（アジャイルコーチ）[15]が参加して手法に不慣れなチームをサポートする．

このモデルを授業に適用することにより，学生は実際のソフトウェア開発に限りなく近い環境で学ぶことができる．

2.5 アジャイル型のソフトウェア開発教育

アジャイルの先進国である米国では，アジャイル型開発をソフトウェア開発教育に取り入れている．

フォートルイス大学では，サービス・ラーニングモデル[b]に基づき，クライアントのためのアプリケーションを開発するプロジェクトを実施している．非アジャイル型の開発手法を採用していた時期は失敗プロジェクトもあったが，アジャイル型を採用後は，すべてのプロジェクトが成功することができた[16]

カーネギーメロン大学の Feng Ji らの研究[17]では，ウォーターフォール型とアジャイル型（XP）の開発手法を比較して，同じ機能を完成させたが，ウォーターフォール型の場合，アジャイル型よりも，多くの時間をコーディング，設計書などのドキュメント作成に費やしたとしている．

また，アジャイルについて PBL の形態で学ぶ．そこでは，産学連携により，実際の顧客が参加すること，顧客とゴールを共有することで，チームのコラボレーションやコミュニケーションの重要性を学ぶ．さらに，欧米ではアジャイルコーチを起用して，技術情報や経験が伝播されている．

2.6 アジャイル型 PBL のプロジェクト評価指標

アジャイル開発では，顧客にとってビジネス価値の高いソフトウェアを生み出すことが望まれる．そのため，アジャイル型 PBL におけるプロジェクト評価指標は，ビジネス価値が高い製品を作れたかという点になる．製品が作れたとしても，顧客満足を得ることができなければ，プロジェクトは失敗である．しかし，実務経験がない学生は，この品質が高い製品についての考えがないため，プロジェクトが失敗するケースがある．

客観的に顧客の好みを理解するためのモデルである狩野モデル[18]によると，品質には，「当たり前品質」と「魅力的品質（期待を超える品質）」の二種類がある．納期通りに納品する，欠陥がないといったことは当たり前品質である．顧客は，当然あるものだと考える．それが提供されないと不満を持つ．顧客に付加価値を提供すること（魅力的品質を提供すること）が必要である．製品の品質が高いと，顧客の総合満足度も高めることができる．狩野モデルを利用した顧客満足度の評価方法は，提案手法で述べる．

3. ソフトウェア品質

3.1 ソフトウェア品質の定義

ソフトウェア品質は，ソフトウェア工学の領域で定義されている．ISO9000 の品質定義では，「品質は顧客の明示的要求と暗黙的要求を満たすこと」として定義されている．

また，ISO/IEC9126（ソフトウェア製品の評価，品質特性とその適用に関するガイドライン）に体系的に整理されている．この指標は，6 種類の品質特性と 21 種類の品質副特性に分類されている．この品質特性の機能性では，「目的から求められる必要な機能の実装度合い」を指すが，アジャイルにおいては，意味が変わる．次に ISO/IEC9126 をアジャイル向けに修正した品質特性を挙げる．これらの品質特性を満たすことがアジャイル型ソフトウェア開発 PBL を成功させるための鍵である．

- 機能性

顧客にとって価値ある機能が実装されている度合い
（無駄な機能の実装は価値がない）

- 信頼性

機能が正常動作し続ける度合い
（障害が発生しても，即座に修復する）

- 使用性（ユーザビリティ）

分かりやすさ，使いやすさの度合い

- 効率性

目的達成のために使用する資源の度合い

- 保守性

保守作業（修正のしやすさ）に必要な努力の度合い

- 移植性

別環境へ移したと際，そのまま動作する度合い

3.2 機能性

アジャイル開発における機能性は，3.1 の通り，仕様書に書かれた機能がシステムに反映されていることではなく，顧客にとって価値ある機能が実装されていることが基準となる．そのため，機能性の評価は顧客満足度と比例する．

3.3 信頼性

システムの障害が少なく安定稼働しているかという基準で信頼性を評価する．信頼性に関しても，アジャイル開発では，障害回復時間（MTTR）を短くすることで，稼働率を高め信頼性を確保する．

近年，クラウドプラットフォームの普及に伴い，AmazonEC2（Amazon Elastic Compute Cloud）やさくらインターネットのレンタルサーバなどの PaaS 環境を本授業で採用している．AmazonEC2 では，99.95%以上のサービス稼働率を保障している[19].

b) サービス・ラーニングモデルとは，米国において実施されている教育手法である．学生が社会貢献活動を通して社会問題に取り組み，学び，成長する．

ⓒ 2013 Information Processing Society of Japan

3.4 保守性

アジャイル開発では，2.3 で紹介したリファクタリングにより，システムの保守性を高めることができる．ソフトウェア開発においては，要求変更や障害対応でプログラムの変更が頻繁に発生する．リファクタリングにより，変更による品質劣化を未然に防ぐ．

3.5 移植性

3.3 で述べたクラウドプラットフォームを利用することで新規にハードウェアを用意して，サーバ環境を構築するより，簡単に移行することが可能である．

4. アジャイル型 PBL の試行

4.1 2011 年度のプロジェクト

2011 年度は，アジャイル型開発手法を取り入れたプロジェクトが 3 チーム結成された[20]．それぞれのチームは試行錯誤しながら，アジャイル開発を進めることになった．

また，作業の見える化や作業効率を上げること目的に，アジャイル型開発プロセスと親和性が高いチケット駆動開発（TiDD: Ticket Driven Development）を採用した．チケット駆動開発を実施するために，バグ管理システム（BTS: Bug Tracking System）である Redmine を導入した．このツールは，バグの管理だけでなく，課題管理システム（ITS: Issue Tracking System）として運用することができる．

しかし，このツールが学生から利用されることは，ほとんどなかった．学生が普段から利用しているツールでないと，導入することが厳しいことが分かった．

4.2 スキルアンケート

次世代ロボットサービスの実現をテーマに，プロジェクトを開始したチームに，スキルアンケートと小テストを実施（表 1）した．

その結果，学生 A の能力が高く，アプリケーション作成の経験もあることが分かった．学生 B はプログラミング能力が低いことが分かった．学生 C はツールやアプリケーション作成の経験はないが，プログラミングの素質が高いことが分かった．

表 1　実施前のスキルアンケート
Table 1　The skill questionnaire before beginning project

質問	学生 A	学生 B	学生 C
プログラミングを授業で習った	○	○	○
基本的な文法を知っている	○	×	○
Eclipse を使うことができる	○	×	×
簡単なアプリを作れる	○	×	×
Android アプリを作れる	○	×	×
小テスト ※	71.4%	28.5%	85.7%

※　IPA の基本情報技術者試験過去問題（Java）

4.3 トレーニングとコードレビュー

スキルアンケートにより，学生らに能力差があることがわかった．そのため，Android アプリケーションのプログラミングができるようにトレーニングを実施した. 学生 B, に対しては，Java の基礎的な文法から学習し，学生 C は Android アプリケーション開発に関して学習した．

プロジェクト終了後に，学生のレポートからトレーニングとコードレビューに関して，学習効果があることがわかった．

学生 C のレポートに着目すると,『Android 開発にあたって欠かすことのできない基礎知識を身につけることができた．また，コードレビューの結果，どうすれば他人にとって見やすいコードになるかということについても学ぶことができた』とコードレビューに関しての効果を読み取ることができた．学生自身は，他人に自分の書いたコードを読んでもらうという経験が皆無であるため，教育的な効果が期待できる．

また，トレーニングに関しては『Android 開発にあたって欠かすことのできない基礎知識を身に付けることができた』と効果があることがわかった．

しかし，多くの時間をトレーニングに費やした学生 B に関しては,『個人の技術が低いとチームに影響させてしまい, 妨げに発展してしまうのかと理解した．技術を習得しながら作業するという活動もまた膨大な時間が必要となり，結局は足を引っ張るという始末に終わってしまった』と述べている．

この点から読み取れるように，学生 C に関してはプログラミングの資質があり，短期間のトレーニングにより，効果的に作業することができたが，学生 B に関してはトレーニングに行っても膨大な時間が必要であり，授業の決められた期間と時間の中で効果的な教育を行うことが困難であることがわかった．また，実際の開発ではなく，トレーニングを割り当てることで学生 B のモチベーションも低下してしまった．

この結果から，ある程度のプログラミングの経験がある場合はトレーニングが有効であるが，授業でプログラミングの構文やアルゴリズムを習っただけの初心者レベルの学生にとっては，プログラミング教育についてトレーニングでは効果があまり望めないことがわかった．

ただし，最後に学生 B は『一人で悩んで時間が過ぎていくより，分かっている人に早く聞いて，作業を進めることが必要』と述べている．チーム内のコミュニケーションが十分に取れていることも必要であった．

4.4 アジャイル開発におけるコードレビュー

一般的にコードレビューは，ソフトウェア開発工程において，早期にバグを発見する目的で実施される．しかし，アジャイル開発ではあまり行われていない．

なぜなら，コードレビューを実施するには，レビュアーの選定，会議室の予約，スケジューリング，資料の印刷，会議の時間を合わせるなどの必要があり，多大な時間を要

ⓒ 2013 Information Processing Society of Japan

してしまう．さらに，多量のソースコードを読まなければならなく，単純な検査（警告，コードスタイル，テストカバレッジなど）に時間を浪費してしまう．

また，Gilb らの研究によると，ピアレビューの場合，レビューできるコード行数は 2 時間で 200 行程度とされている[21].

以上のことから，アジャイル型のような軽量型開発手法の場合，コードレビューをそのまま実施することは難しいとされている．授業に参加しているアジャイルコーチの吉羽龍太郎氏は，アジャイル開発におけるコードレビューは『ソースコードが少量のうちから頻繁に実施するのが基本思想であり，問題は小さいうちに解決した方がよい』と述べている．この知見から，通常のアジャイル開発においてソースコードに変更があった場合は，即座にコードレビューできる環境が必要であると言える．

4.5 プラクティスとしてのコードレビュー適用

アジャイル開発において，品質に影響を与えるプラクティスは，2.3 で紹介したテスト駆動開発，ペアプログラミング，継続的インテグレーション，リファクタリングなどである．しかし，それらプラクティスの導入は学生とって敷居が高い．

ペアプログラミングをソフトウェア開発 PBL に適用した場合，同じ箇所で作業することが前提となる．しかし，時間が決められた授業でプロジェクトを実施する場合は，学生は授業時間外に各自作業することになる．

2011 年度のプロジェクトデータでは, 総実績工数が 429.5 時間であった．その内の 225 時間が講義やミーティングを含む時間である．そして，204.5 時間は授業外での作業時間となる．また，学生が授業を欠席する場合もあるため，授業時間内だけでペアプログラミングにより作業を行うことは難しい．

そのため，分散開発においてレビューができる環境が必要である．表 2 に Wojciech.Seliga らが提唱するコードレビューとペアプログラミングの比較をまとめる[22].

コードレビューを採用する利点は，導入する敷居が低いこと, 非同期かつ, 分散開発が可能である点が挙げられる．

分散環境において，ペアプログラミングを実現する研究はされているが，時間的制約やレビュアーのプログラミング能力が考慮されていない．

表 2　コードレビューとペアプログラミングの比較

Table 2　Comparison of a code review and pair-programming

コードレビュー	共通	ペアプログラミング
目的は検証	知識の共有	目的は創作
非同期	品質向上	同期
分散開発可	共同作業	分散開発不可
敷居が低い		敷居が高い
広範囲		集約的

5. 関連研究

5.1 品質特性駆動アジャイル開発

S.Jeon らは，アジャイル開発における失敗プロジェクトは不十分な機能ではなく，メンテナンスの困難さ，低いパフォーマンス，セキュリティ問題のようなソフトウェア品質特性が問題であるとしている[23]．そして，以下の問題点を挙げている．

- バックログは機能（フィーチャー）のみに着目しているため，品質特性を反映することが難しい
- バックログの追跡可能性（トレーサビリティ）を分析しないで，実装に注目しているため，要求変更が生じた場合，プロジェクトを維持することが難しい
- 多人数のチームで実践することが難しい

上記の問題に対し，機能性，有用性，信頼性のような品質特性を考慮した品質特性駆動のアジャイル開発手法「ACRUM(Attribute-driven SCRUM)」を提案している．

彼らの提案手法では，バックログに対して品質特性を考慮して，ソフトウェア品質に対する補完を行っているが，本手法では，コードレビューの支援環境によりソフトウェア品質を向上させる．

5.2 Continuous Changeset-Based Review (CCBR)

Winkler らの研究[24]によると，アジャイル開発ではペアプログラミングとインスペクションを同時に行うと欠陥検出率が上昇するとしている．

しかし，時間を消費するため，Mario.Bernhart らは，サポートする手法として CCBR を提案している[25]．CCBR では，リポジトリの内容変更時にリアルタイムにコードレビューを行う仕組みである. 表 3 のように, 実装者に対して, 経験豊富なレビュアーが割り当てられる．この手法では，ファイルの拡張子をフィルタリングして，適切なレビュアーに割り当てられるように工夫をしている．

レビュアーは，比較ツール（図 2）を利用してファイルの差分からレビューを実施する．比較ツールには前バージョンとの差分が表示される．レビュー後に結果を実装者にフィードバックする．

また，提案手法を空港のオペレーションソフトウェア開発に適用して，有用性，効率性を評価した[26].

表 3　CCBR におけるレビュアーの選定

Table 4　Selection of reviewer in CCBR

ID	Review Assignments		
	Author	Reviewer	Filter
1	Peter	Mike	*.java; *.xml
2	John	Mike	*.java; *.xml
3	Ben	Mike	*.java; *.xml
4	*(all Authors)	Chris	*Test*.java

Mike: senior developer

Chris: test expert

ⓒ 2013 Information Processing Society of Japan

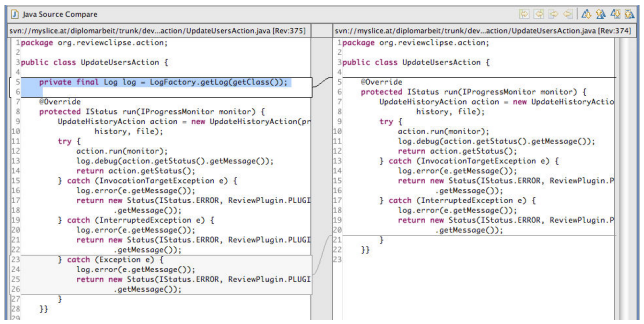

図 2　比較ツールの利用
Figure 2　Use of a comparison tool

5.3 CCBR を PBL に適用した場合の問題

5.2 で紹介した Mario.Bernhart らの提案手法（CCBR）をアジャイル型ソフトウェア開発 PBL に適用した場合を考察する．

既存手法の CCBR の場合，レビュアーは経験豊富なエンジニアである．しかし，ソフトウェア開発 PBL の場合は，レビュアーも学生のため，実務経験を持たない．そのため，どこが悪いのかレビュアーもわからない，コードレビューの優先度がわからない．という問題が発生し，指摘できる箇所も不十分となってしまう．

6. 提案手法

6.1 継続的インテグレーション（CI）支援ツールの活用

初めに 2.3 で紹介した CI を利用した開発支援環境を提案する（図 3）．本システムは，GitHub（プロジェクトホスティングサービス）[27]，Jenkins（CI ツール），Apache Ant（ビルドツール），xUnit（テスティングフレームワーク），PMD（静的解析ツール）[28]を利用している．これらを組み合わせて，CI とリファクタリングの環境を構築した．リファクタリングは，CI と組み合わせることで効果が上がる．

静的解析を行うことで，短い変数名やメソッド名を検出したり，複雑度の高いコードを検出するなどバグになりやすい箇所を見つけたり，保守性を高めることができる．

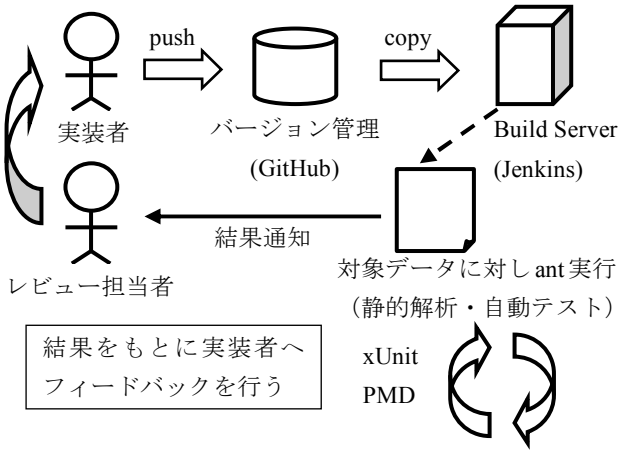

図 3　CI を利用した開発支援環境
Figure 3　Development support environment using CI

6.2 分散型バージョン管理システムの利用

GitHub とは，分散型バージョン管理システム Git をサポートするためのツールである．近年，分散型でプログラムなどのソースコード管理を行う Git が利用されている．Subversion などの集中型バージョン管理システムと異なり，インターネットに接続されていなくてもローカル環境において作業することが可能である．

また，GitHub はリモートの公開リポジトリ（リモートリポジトリ）であり，ドキュメント生成機能や Wiki 機能トラッキング機能など様々な機能で Git による分散管理を補助することができる．リモートリポジトリとは，インターネット上に存在するプロジェクトのことである．複数のリモートリポジトリを持つことも可能であるし，それぞれを読み込み専用にしたり，読み書き可能にしたりすることもできる．

他のメンバと共同作業を進めるには，リモートリポジトリを管理し，必要に応じて，データのプル（pull），プッシュ（push）を行うことで作業を分担する．具体的には図 4 の流れとなる．

1. インデックス（ステージ）にコミット対象のファイルを登録する
2. コミットして，ローカルのリポジトリに，インデックスの変更を確定させる
3. プッシュして，リモートリポジトリに，ローカルの変更を適用する

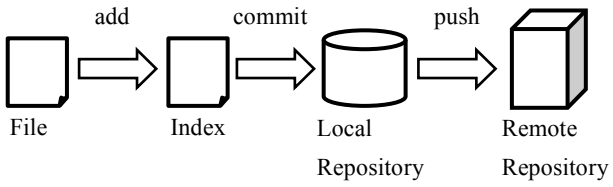

図 4　リモートリポジトリの利用
Figure 4　Use of a remote repository

6.3 CI ツール

Jenkins とは，CI を実現するためのツールである．リポジトリは，Git を利用している．Jenkins が配置されているサーバーから常時，Git をポーリング（監視）しているため，対象のリモートリポジトリに変更があった場合，変更情報が検知され，対象データがサーバーにプル（コピー）される．

そして，ビルドツール（Apache Ant）を動作させ，自動テスト（xUnit）と静的解析（PMD）を行う．結果はレポート出力させる．結果はレビュー担当者に通知され，実装者へフィードバックする際に役立つ．CI ツールを活用することで，レビュー担当者への支援と効果的なレビューによる指摘が期待できる．それにより実装者への教育的効果も向上する．

ⓒ 2013 Information Processing Society of Japan

6.4 CCBR の拡張

CI 環境を利用して, 既存手法の CCBR をソフトウェア開発 PBL 向けに拡張した. 本手法のワークフローを以下に示す．追加部分はワークフローの 2, 3, 4, 6, 8, 10 である．

- ワークフロー

1. 学習者がリポジトリにコミット
2. テスティングツール，静的解析ツールによるチェック
3. レビュアー候補を選定

> 選定方法:一定時間のバグ検出数が最大の学習者をレビュアー候補とする

4. テスト結果，解析結果のレポート出力
5. レビュアーとマッチング
6. 狩野モデルによる優先度付け
7. レビュアーに前バージョンとの差分を表示
8. テーラリングによる観点の設定
9. レビュアーがレビュー実施，結果をフィードバック
10. レビュー結果により，学習者がリファクタリングする

6.5 静的解析ツールによるコードレビュー局所化

静的解析ツールとは，ソースコードを解析してコーディング規約違反や潜在的にバグになりそうな箇所を検出するツールである．

本システムでは，PMD を採用した．FindBugs などと異なり，既存のルールだけでなく，新たにルールを作ってチェックすることが可能である．ソースファイルを構文木に分解し，その木構造をたどることで規約違反箇所を検索する．そして，未使用の変数や，不必要なオブジェクトの作成などを発見することができる．使用方法は，ビルドツールから実行する．静的解析ツールの導入は，コードレビューをサポートすることが目的である．

6.6 狩野モデルによるレビュー箇所優先度付け

2.6 で紹介した狩野モデルによるレビュー箇所の優先度付けを行う．プロダクトオーナーの主観や思惑抜きで要件（フィーチャー）の優先度が決まるため，このモデルは合理的である．

開発対象となる機能が実現された場合（機能設問）と実現されなかった場合（逆機能設問）について 5 者択一の質問をし，評価二次元表（表 5）から優先順位付けを行う手法である．設問の回答は以下のようになる．

1. 気に入る（Like）
2. 当然だと思う（Expect）
3. 何とも思わない（Neutral）
4. 別にそれでも構わない（Like with）
5. 気に入らない（Dislike）

例えば，機能設問が「当然だと思う」，逆機能設問が「気に入らない」場合，評価二次元表より Mandatory が優先順位となる. Mandatory は, なくてはならないものを意味する．また，Linear は，なくてはならないが，あればあるほど良いもの．Exciter は，差別化のポイント．Questionable は回答がおかしいこと．Reverse は，あっては困るもの．Indifferent は，どうでもいいものを表している．

表 5　狩野モデルにおける評価二次元表

Table 5　The two-dimensional table in a Kano model

機能設問 ＼ 逆機能設問回答	1	2	3	4	5
1	Q	E	E	E	L
2	R	I	I	I	M
3	R	I	I	I	M
4	R	I	I	I	M
5	R	R	R	R	Q

M: Mandatory（基本）
L: Linear（性能）
E: Exciter（魅力）
Q: Questionable（不明）
R: Reverse（逆行）
I: Indifferent（不問）

6.7 コードレビュー優先度

6.6 より決定された優先度の高い機能（フィーチャー）からコードレビューを実施する．優先度はスプリントバックログに記載されている（表 6）.

例えば優先度の評価が Mandatory である機能が満たされない場合は，顧客満足度が低下するため，順次開発を進める．Liner や Exciter の場合は，顧客に対して，期待を超える品質を提供できるためスプリント期間に実装できるとよい．Questionable，Reverse，Indifferent は，そもそも実装しないのでコードレビューをする必要もない.

表 6　スプリントバックログ

Table 6　Sprint backlog

ID	フィーチャー	優先度	担当者
#1	開発環境を構築する	-	全員
#2	Web サーバーを構築する	-	学生 A
#3	新機能 A を作る	Mandatory	学生 B
#4	機能 B に機能を追加する	Liner	学生 C
#5	機能 C に機能を追加する	Exciter	学生 D

6.8 テーラリングによるレビュー観点の設定

テーラリングとは，プロジェクトの特性や事情を考慮し，インスペクションをカスタマイズすることである[25].

既存手法では，技術者のスキルによりレビューの観点を設定するが，ソフトウェア開発 PBL では，学生が初級プログラマかそれ以下である場合が多い．そのため，観点の設定はすべて初級にする．

また，既存手法はユースケースをもとにレビューを行う．提案手法では，スプリントバックログのシナリオをもとにレビューする．これにより，プロダクトオーナーが求めている機能の品質を高めることができる．

6.9 狩野モデルの価値基準を利用した顧客満足度の決定

アジャイル型ソフトウェア開発 PBL における顧客満足度を次の式で定義する．従来の顧客満足度の指標は，アンケートによる評価が一般的であった．提案手法では，顧客満足度を定量的に評価する．

ⓒ 2013 Information Processing Society of Japan

また，顧客満足度の指標はソフトウェア品質特性の機能性の度合いと同義の意味を持つ．顧客満足度の求め方であるが，プロダクトオーナーが作成したプロダクトバックログの要求項目より，狩野モデルから必要であると定義されたフィーチャーの合計を分母に設定する．また，分子に対象フィーチャーに進捗度をかけた値を設定する．そこから，実現されたフィーチャーの割合（%）を計算する（図 4）．対象フィーチャーの M, L, E は，それぞれ，Mandatory（基本），Linear（性能），Exciter（魅力）を表す．

$$CS = \frac{\sum_{i=1}^{n}(\alpha_i \times P_{prog})}{\sum_{i=1}^{n}\alpha_i} \times 100$$

顧客満足度：CS
対象フィーチャー：α
対象フィーチャーの合計：n
進捗率：P_{prog}

図 4　顧客満足度の定義
Figure 4　Customer satisfaction

7. おわりに

本論文では，アジャイル開発におけるコードレビュー手法 CCBR を拡張した．ソフトウェア品質特性に着目し，CI ツールとの連携，静的解析ツールによるコードレビューの局所化，狩野モデルによる優先度付け，顧客満足度の定量的な評価方法を提案した．

本研究は，アジャイル型ソフトウェア開発 PBL における学習者のコードレビュー支援を目的とした．実社会のコードレビューは，レビュアーは経験豊富なエンジニアが担当し，経験の少ないエンジニアを指導することが適切である．コードレビューにより，ソフトウェア品質を高めると同時に開発スキルの向上を図ることができる．PBL では学生同士がレビューすることが前提となる．

しかし，実務経験がない学生の場合，レビュアーの選定が難しい．また，何を基準とするのか，どこからレビューするかわからないといった問題が起こり，コードレビューを実施しても指摘できる箇所も不十分となってしまう．

謝辞　本研究は慶應義塾大学における産学連携プロジェクト「協創型ソフトウェア開発」の中で実施しました．産業技術大学院大学の中鉢欣秀先生，慶應義塾大学の岡田健先生，アジャイルコーチの皆さま．そして，プロジェクトに参加した学生の皆さまに多大なご協力をいただきここに謝意を表します．

参考文献

1) Davenport, D.: Experience Using a Project-Based Approach in an Introductory Programming Course, IEEE Trans. Education, Vol.43, No.4, pp.443-448 (2000)
2) 沢田篤史, 小林隆志, 金子伸幸, 中道上, 大久保弘崇, 山本晋一: 飛行船制御を題材としたプロジェクト型ソフトウェア開発実習, 情報処理学会論文誌 Vol.50, No.11, pp.2677-2689 (2009)
3) 井垣宏 他: 実践的ソフトウェア開発演習支援のためのグループ間比較にもとづくプロセスモニタリング環境, 日本教育工学会論文誌 34(3), pp.289-298, (2010)
4) 松澤芳昭, 大岩元: 産学協同の Project-based Learning によるソフトウェア技術者教育の試みと成果, 情報処理学会論文誌 Vol.48 No.8 pp.2767-2780 (2007)
5) 松澤芳昭, 杉浦学, 大岩元: 産学協同の PBL における顧客と開発者の協創環境の構築と人材育成効果, 情報処理学会論文誌 Vol.49 No.2 pp.944-957 (2008)
6) Satoru Kizaki, Cyubachi Yoshihide: Improvement of collaborative work on software development PBL in a distributed environment, Invitation to 3rd International PBL Symposium 2012 (2012)
7) Jeffrey Hammond: Five Ways To Streamline Release Management, Forrester (2011)
8) Kent Beck, B. Boehm: Agility through discipline A debate, IEEE Computer, pp.44-46 (2003)
9) Linda Rising, Norman S. Janoff: The Scrum Software Development Process for Small Teams, IEEE Software, pp.26-32 (2000)
10) H. Takeuchi and I. Nonaka: The new new product development game. Harvard business review, Vol. 64, No. 1, pp. 137-146 (1986)
11) Esther Derby, Diana Larsen and Ken Schwaber: Agile Retrospectives: Making Good Teams Great (2006)
12) Kent Beck: Test-Driven Development: By Example. Addison-Wesley Professional. pp. 240 (2002)
13) Alan Shaw, Ph.D.: Extending the Pair Programming Pedagogy to Support Remote Collaborations in CS Education, 6th International Conference on Information Technology: New Generations, pp.1095-1099 (2009)
14) Jenkins, http://jenkins-ci.org/
15) Silva, K.; Doss, C.: The Growth of an Agile Coach Community at a Fortune 200 Company, Agile Conference (AGILE), pp.225-228 (2007)
16) Hanks, B.: Becoming Agile using Service Learning in the Software Engineering Course, Agile Conference (AGILE) 2007, pp.13-17 (2007)
17) Feng Ji: Comparing extreme programming and Waterfall project results: Software Engineering Education and Training (CSEE&T), 2011 24th IEEE-CS Conference on, pp.482-486 (2011)
18) 狩野紀昭, 瀬楽信彦, 高橋文夫, 辻新一, 「魅力的品質と当たり前品質」, 『品質』, 14, No.2 pp39-48 (1984)
19) Amazon Web Services AmazonEC2 サービスレベルアグリーメント，http://aws.amazon.com/jp/ec2-sla/
20) 木崎悟，丸山英通，土屋陽介，中鉢欣秀: ソフトウェア開発 PBL へのチケット駆動開発の適用による共同作業の改善，2011 年度プロジェクトマネジメント学会秋季大会 (2011)
21) T. Gilb, D. Graham, and S. Finzi.: Software Inspection, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (1993)
22) Wojciech.Seliga, Slawomir.Ginter: Effective Code Review in Agile Teams, 2009 Agile Conference (2009)
23) Sanghoon Jeon, Myungjin Han, Eunseok Lee, Keun Lee: Quality Attribute Driven Agile Development, 9th International Conference on Software Engineering Research, Management and Applications (2011)
24) D.Winkler and S. Biffl.: An empirical study on design quality improvement from best-practice inspection and pair programming, 7th international conference on Product-Focused Software Process Improvement, pp.319-333 (2006)
25) Mario Bernhart, Andreas Mauczka, Thomas Grechenig: Adopting Code Reviews for Agile Software Development, 2010 Agile Conference (2010)
26) Mario Bernhart, Stefan Strobl, Andreas Mauczka, Thomas Grechenig: Applying Continuous Code Reviews in Airport Operations Software, 12th International Conference on Quality Software (2012)
27) GitHub, https://github.com/
28) PMD, http://pmd.sourceforge.net/

ⓒ 2013 Information Processing Society of Japan