アジャイルプロジェクトにおけるペアワーク適用の改善事例

Improvement of the pair work application in the agile project

日本電気株式会社 ソフトウェアエンジニアリング本部
NEC Corporation, Software Engineering Division
○小角　能史　　佐藤　孝司 1)
○Yoshifumi Kosumi　　Takashi Sato1)

Abstract In recent years, the agile software development method is widely applied to projects to flexibly and rapidly respond to a request from business. In agile projects, it is important to deliver working software rapidly. Because requirements, which are divided into smaller units, are developed from the most important one, the project owner can get software which respond to changes in business requirements flexibly and rapidly. Various practices are used to obtain such benefits in agile projects.
Pair work is one of the agile practices. As the application rules of pair work is customized in accordance with the project's properties, the development team can eliminates oversights and shares information. When customizing the application rules of pair work, it is important to optimize with viewpoints such as the scope of application, how to create pairs, and reshuffling pairs. Adding rules for reviews effectively enhances information-sharing.
This paper presents a case in which the projects has optimized the rules for applying pair work through retrospectives held at the end of each sprint. It describes the resultant rules for applying pair work. It suggests that the rules can be effectively applied to other projects.

1. はじめに

近年、ビジネス領域からの要求に柔軟かつ素早く対応するために、ソフトウェア開発においてアジャイル開発の手法が広く採用されている。アジャイル開発では、動作する成果物を素早く届けることを重視しており、小さな単位に分割されたソフトウェアの開発要件を重要なものから開発することで、ビジネス要求の変化に柔軟かつ迅速に対応可能というメリットが得られる[1]。

アジャイル開発では、このメリットを得るために様々なプラクティスが存在している。これらのプラクティスはただ実行するだけで効果が得られるわけではなく、プロジェクトの条件に合わせて各プラクティスの適用方法を工夫することで効果を得ることができる。

ペアワークはアジャイル開発のプラクティスの 1 つである。ペアワークでは 2 人で会話をしながら調査や設計、製造、テストを実施する[2]。ペアワークを適用することで、開発者が個別に作業するよりも作業中の見落としが少なくなる。さらに、ペアを時々ローテーションすることで、開発者はお互いに訓練し合ったり、情報を共有したり、信頼関係を構築したりできる[3]。これらの効果を得るために、筆者が所属する NEC では、アジャイル開発プロジェクトにおいては、ペアワークを利用することを強く推奨している[4]。しかし、ペアワークの具体的な適用方法はプロジェクトごとにその条件にあわせて作成する必要があり、ペアワークを適用した経験がない場合には、その作成が困難である。

本論文では、変化するビジネス要求に柔軟かつ迅速に対応するというアジャイル開発のメリットに対して、ペアワークを利用して効果的に成果を出すために、初期のペアワーク適用ルールを作成し、スプリント終了時にふりかえり[5]を実施することで適用ルールを最適化した事例を紹介

日本電気株式会社 ソフトウェアエンジニアリング本部
Software Engineering Division, NEC Corporation
東京都港区芝 5 丁目 7−1　Tel: 03-5378-9813　e-mail: y-kosumi@cp.jp.nec.com
5-7-1 Shiba Minato-ku Tokyo Japan

1) 日本電気株式会社　生産技術・品質保証本部 主席主幹
Exective Specialist, Production Engineering and Quality Promotion Division, NEC Corporation

する。その結果得られたペアワークの適用ルールを示し、この適用ルールの他プロジェクトでの利用可能性について述べる。

2 章でペアワーク適用ルールを作成するために考慮するべき観点を説明する。3 章で実際のプロジェクトにペアワークを適用して改善した事例を示す。4 章では事例で得た適用ルールの有効性について考察し、5 章で本論文の結論を述べる。

2. ペアワーク適用ルール作成における課題

ペアワークを利用して、ビジネス要求の変化に柔軟かつ迅速に対応するためには、プロジェクトの条件に最適化された適用ルールが必要である。適用ルールを最適化する際には開発者のスキルなど、プロジェクトの特性を考慮する必要がある。

一方、初めてペアワークを適用するプロジェクトにとっては、適用開始時にプロジェクトの特性を考慮して初期のルールを作成すること自体が課題となる。例えば、実際にペアワークを適用する際には少なくとも以下のような観点に関するルールを作成する必要がある。

- ペアワークの適用範囲
- ペアの組み方
- ペアの定期的な交代（ローテーション）の有無

表 1　プロジェクトの概要

項目	内容
アジャイル開発方法論	スクラム
開発者の人数	4 名
開発期間	1 年 10 か月
1 スプリントの期間	2 週間
開発対象ソフトウェアの特徴	● データベースを検索し、グラフ表示などを行う社内向けの Web システム ● 開発の難易度は平均的 ● V1.0（初版）の開発ではペアワークを適用せず、V2.0 からペアワークを適用
ドキュメントの作成	以下の目的で設計書を作成 ● 機能間の整合性確認 ● 将来の保守用

ペアワークの適用経験が無い開発者にとっては、どのような基準で上記の内容を判断すればよいかわからない可能性がある。そこで筆者らは、ペアワークを適用しない場合の開発経験を参考に最低限必要なルールを作成して、それを改善することでプロジェクトの特性に最適化されたペアワーク適用ルールを得ることを考えた。

3. プロジェクトでのペアワーク適用ルールの最適化

3.1 対象プロジェクトの概要

表 1 に示す条件のプロジェクトにペアワークを適用し、ペアワークの効果的な適用ルールについて検証した。

このプロジェクトでは、アジャイル開発の方法論としてスクラムを適用した[6]。スクラムでは、プロダクトバックログと呼ばれるプロダクト要件の一覧を元に、スプリントと呼ばれるイテレーションを繰り返してソフトウェアを開発する。スプリントでは図 1 のとおり、最初に行うスプリント計画でプロダクトバックログから優先順位の高い要件を選択して、各要件を開発するための設計や製造、テストなどのタスクに分割する。スプリント期間中はデイリースクラムと呼ばれる短いミーティングで毎日開発者間の情報共有を行いながら、タスクの単位で開発を行う。スプリントの終わりには、スプリントレビューで動作する成果物を確認し、さらにふりかえりを行うことでプロジェクト遂行中に気付いた課題を

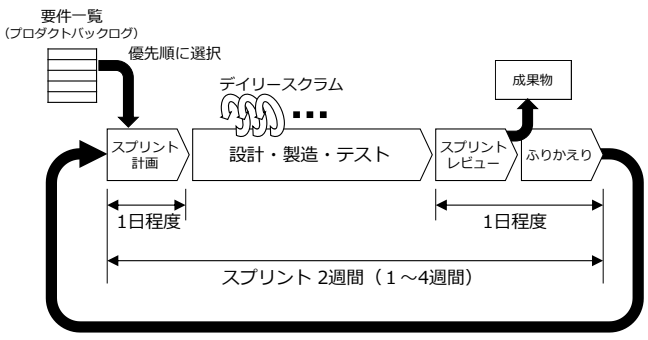

図 1 スクラムのプロセス

改善する。複数回のスプリント実施後にリリースの基準を達成していると判断できた時に、そのソフトウェアをリリースする。

対象プロジェクトでは、毎回のスプリント終了時にふりかえりを実施し、プロジェクト遂行中に気付いた課題を改善した。ふりかえりにはKPT[7]を用い、開発者全員でスプリント期間中に感じた問題に対する解決策を立案した。KPTとは、Keep（今後も維持したいこと）、Problem（課題）、Try（改善したいこと）という視点で情報を整理して、プロセスを改善する手法である。

3.2 適用開始時のルール

V2.0の開発からペアワーク適用開始するにあたり、対象プロジェクトでは初期のペアワーク適用ルールとして表 2に示すルールを作成した。表 2に示した各観点と適用ルールを作成する際の考え方について、以下にそれぞれ説明する。

表 2 初期のペアワーク適用ルール

観点	適用ルール
適用範囲	● 調査、設計、プログラム作成、テスト項目作成、テストプログラム作成にはペアワークを適用する ● テスト実施、スパイク（技術検証のための試作）は適用対象外とする
ペアの組み方	● ペアのいずれかが作業を実施するのに必要なスキルを保有している ● ペアのいずれかが改造前の機能について設計やプログラムを理解している
ペアのローテーション	● 可能であれば1日1回ペアをローテーションする

（1） 適用範囲

ペアワークは設計やプログラム作成を行うドライバーと、ドライバーの作業を常時確認するナビゲーターがペアとなり作業を行うプラクティスである。ペアワークではナビゲーターが常時成果物を確認することで設計やプログラム中の誤りを摘出し、その場で修正する。これは、常時レビューをすることに相当する。

そこで、ペアワーク適用開始前のバージョンにおいて、レビュー対象となる成果物を作成していたすべての作業をペアワークの適用範囲とした。

（2） ペアの組み方

ペアワークにより作業中の見落としを少なくするには、作業実施に必要な知識・スキルが必要

表 3 KPTで指摘されたProblemおよびTryと改善結果

観点	Problem	Try	改善結果
適用範囲	テストプログラムの製造では、ナビゲーターは指摘する機会がほとんどない	テストプログラムの製造はペアワーク適用外に変更	ナビゲーターの待ち時間を削減
ペアの組み方	メンバ変更時に、新メンバは他のメンバと同等の開発ができない	新メンバのペアの相手は、対象作業をもっとも理解したメンバを優先的に割り当てる	新メンバは1回のスプリントで必要なスキルを習得できた
ペアのローテーション	毎日のローテーションができず、設計や実装の情報共有ができない	毎朝ペアを必ず変更することにし、仕掛中のタスクを継続するペアはそれまでの担当ペアの1人を含める	毎日ペアが変更され、開発チーム内での知識の共有が進んだ
レビュー	他ペアが作成した設計を把握していないことが原因の後戻りが発生	外部設計は開発者全員参加のレビューを実施するようにルールを変更	設計に関する知識を共有できるようになり、仕様の不整合を防止できるようになった
休憩の取り方	ペアワーク中に休憩を取ることを提案しにくく、疲労がたまる	約1時間の作業に対して10分の休憩を取ることをルール化	適切に休憩が取れるようになり、開発者の疲労が緩和された

表 4　改善後のペアワーク適用ルール

観点	適用ルール
適用範囲	● 調査、設計、プログラム作成、テスト項目作成にはペアワークを適用する ● テストプログラム作成、テスト実施、スパイクはペアワーク適用対象外とする
ペアの組み方	● ペアのいずれかが作業を実施するのに必要なスキルを保有している ● ペアのいずれかが改造前の機能について設計やプログラムを理解している ● 特に、新メンバがいる場合は、新メンバとペアを組む開発者は必要なスキルの熟練者とする
ペアのローテーション	● 仕掛中のタスクの有無にかかわらず、毎日ペアをローテーションする ● ペアローテーション時に仕掛中のタスクがある場合は、1 人はそのタスクを継続実施する
レビュー	● 外部設計の成果物は必ずチームレビューを行う
休憩の取り方	● 1 時間に 10 分を目安に休憩を取る ● 休憩したい場合は、自由に休憩を提案してよい

となる。そこで、ペアのいずれかがそれらを持つことをペアの組み方に関するルールとした。

（3） ペアのローテーション

ペアワーク適用時に定期的にペアをローテーションすることで、開発チーム内で成果物に対する知識が共有できることが知られている。このプロジェクトでも、その効果を得るためにペアを定期的にローテーションすることとした。まずは毎日 1 回ペアをローテーションすることを基本とし、具体的には 2 つのペア間でタスクの終了が同期したタイミングでローテーションすることにした。

3.3　ふりかえりによる改善

対象プロジェクトの各スプリントの終わりに、ルール改善ために KPT によるふりかえりを実施した。KPT とは、Keep（今後も維持したいこと）、Problem（課題）、Try（改善したいこと）という視点で情報を整理して、プロセスを改善する手法である。ふりかえりでは Problem の指摘や Try の立案を含めて、すべて実際に開発を行っている開発メンバが中心となり実施した。

ふりかえりで指摘された Problem と対応する Try および改善結果を表 3 に示す。また、その結果得られた改善後のペアワークの適用ルールを表 4 に示す。プロジェクト開始時には、表 2 に示した適用範囲、ペアの組み方、ペアのローテーションという観点に関するルールしかなかったが、実際にプロジェクトに適用することで、レビューや休憩の取り方という新たな観点の課題が見つかり、課題を解決するための適用ルールを追加している。

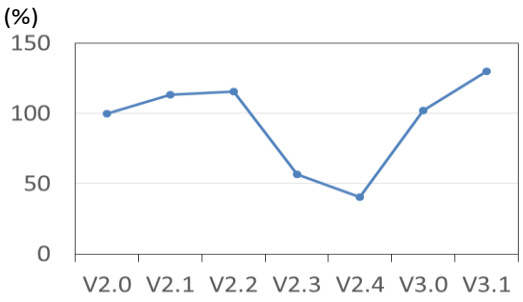

図 2　生産性の変化

表 5　各バージョンの条件

Version	スプリント回数	新規機能の割合	メンバ交代
V2.0	10 回	68%	-
V2.1	4 回	67%	-
V2.2	4 回	60%	あり
V2.3	5 回	37%	-
V2.4	6 回	41%	-
V3.0	7 回	94%	あり
V3.1	4 回	72%	あり

3.4　プロジェクトの生産性実績

プロジェクトの生産性の変化を図 2 に、各バージョンの条件を表 5 に示す。生産性の数値は LOC/人 H で計算した結果に対して、V2.0 が 100%となるように正規化したものを表している。

図 2 では、V2.3 および V2.4 を除き、V2.0 と同等以上の生産性を示している。V2.3 と V2.4 で

は生産性が V2.0 と比べて 50%程度に低下しているが、新規機能の割合が 50%未満で既存機能の改造が中心となり、少量の修正に対しても既存機能のテストを十分に実施する必要があったことから相対的に生産性が低下したものと考えられる。

プロジェクトでは V2.2 の開始時、V3.0 の開始時、V3.1 の途中のそれぞれで 1 名ずつのメンバ交代が発生したが、いずれのバージョンにおいても生産性の低下は見られなかった。また、V2.2 以降ではふりかえりにてペアワークの適用ルールに関する Problem は指摘されておらず、ペアワークの適用ルールは変更されていない。

4. 考察

4.1 要件変化への柔軟な対応

アジャイル開発のメリットである、ビジネス要求の変化に対して柔軟かつ迅速に対応可能な状況を維持するには、プロダクトバックログの中で、優先度が高い要件から順に開発対象を選択することが原則である。

スプリントの計画時に、開発要件を分解した各タスクに対して開発者をアサインするにあたって、タスクに必要な知識やスキルに担当者間で偏りがあると、特定の担当者に負荷が集中し、要件の優先度を見直すことにもなりかねない。逆に知識やスキルが十分でない担当者をアサインすると既存機能の調査から作業することとなり、生産性を低下させてしまう。

3 章の事例のプロジェクトでも、プロダクトバックログ内の要件の優先度は頻繁に変化した。しかし、スプリント中にタスクを開発者にアサインする際に、開発チーム内で既存機能の設計やプログラムなどの知識が共有された状態であったため、要員のアサインに悩む場面はなかった。

4.2 開発メンバの交代への対応

開発チームのメンバが交代する場合、前担当者しか知らない設計やプログラムに関する情報があれば、新担当者に引き継ぐ作業が必要になる。これはプロジェクトにとってリスクであり、十分な引き継ぎが行われなければ、チームとしての生産性を落とすだけでなく、品質の低下を招くことにもなりかねず、ビジネス要件に柔軟かつ迅速に対応することの妨げとなる。

すでに述べたように、3 章の事例プロジェクトでは、V2.2、V3.0、V3.1 のそれぞれで開発者 1 人の交代が発生した。それぞれのタイミングにおいて、前担当者のみが保有する情報の有無を確認したところ、前担当者が持っている情報は他の開発者と共有できていることが確認できたため、前担当者が引き継ぎ作業を実施する必要は無いと判断した。その結果、前担当者は交代の直前まで通常の開発を実施することができた。これはアジャイル開発のプラクティスであるコードの共同所有が、ペアをローテーションすることで自然に実現できたと考えられる。

また、このプロジェクトでは交代で参加した新担当者の立ち上げにも、ペアワークが効果的であった。新担当者は、元からプロジェクトに参加していた開発者とペアを組むことで、既存機能の設計やプログラムを理解し、プロジェクトで採用している技術を習得できた。新担当者はスプリントを 1 回完了した時点で、ペアワークを適用するという条件において、他のペアと遜色なく開発できる状況になった。その結果、図 2 に示すとおり、開発者の交代が発生したバージョンにおいて、それまでのチームとしての生産性を維持することができた。

4.3 他プロジェクトへのルールの適用

表 4 で示した適用ルールは、ペアワーク適用開始時のルールとしては、ある程度汎用的に広い範囲のプロジェクトで有効と考えている。実際のプロジェクトに適用開始した後は、そのプロジェクトの特性に応じて、最適なルールに見直すことが必要である。

表 4 のルールを他のプロジェクトに適用するにあたり、注意するべきポイントとして、適用範囲とペアのローテーションについて以下に補足する。

(1) 適用範囲

表 4 で適用対象外とした作業においても、その作業の難易度が高い場合や開発者間のスキルにバラツキがある場合には、最初はペアワークを適用し、習熟した後に適用対象外に変更するなどの工夫が必要である。こうすることで、開発者間の当該作業に関するスキルを効率的に向上させることができる。

（2） ペアのローテーション

3 章の事例プロジェクトでは、開発者は 4 人と少なく、ペアのローテーションによる情報共有は、比較的実施しやすい条件であった。

ここで、より開発者の人数が多い場合へのルールの適用について考えてみる。スクラムガイドによると、開発チームの人数は小回りが利く程度に小さい必要があり、最大でも 9 人までが理想とされている[8]。9 人を超えてしまうと、開発者間の調整の機会が多くなり、管理も複雑になってしまう。これを踏まえて、ここでは全ての開発者がペアを組むことを前提に、開発チームが 8 人で構成される場合を考える。

対象ソフトウェアのモジュール構成などに応じて、開発者をいくつかのグループに分ける場合や、そのまま 8 人のグループで開発を進める場合が考えられる。

前者の場合は、例えば 4 人ずつの 2 つのグループを構成すれば、3 章の事例と同じ条件であるため、表 4 に記載のペアのローテーションに関するルールがそのまま適用できる。

後者の 8 人のグループのままでペアをローテーションさせる場合は、スプリント期間が 2 週間であれば、各開発者はその他全ての開発者と 1 回以上はペアを組むことができるため、ペアのローテーションによる情報共有の効果を得ることは可能と考えらえる。

5. おわりに

本論文では、実際のプロジェクトにペアワークの適用ルールを最適化した事例と、その結果得られた適用ルールを紹介した。また、ペアワークを初めて適用するプロジェクトが、そのルールを適用することで、効率的にペアワークの適用を開始できる可能性について考察した。

紹介した事例は比較的規模が小さく開発者も 4 人と少ないため、ペアワークによる効果を得やすかったと考えている。しかし、規模がより大きいという条件において、効果的にペアワークを適用するためには、今回は気づかなかった課題が見えてくる可能性がある。

今後は規模や人数といった条件の異なるプロジェクトに対して、本論文で示したルールを適用することで、プロジェクトの条件ごとにどのようなペアワークの適用ルールが適しているかを確認したい。

参考文献

[1] Mike Cohn, アジャイルな見積りと計画づくり, 毎日コミュニケーションズ,2009
[2] ケント・ベック, XP エクストリーム・プログラミング入門―変化を受け入れる（第 2 版）, ピアソンエデュケーション, 2005
[3] James O.Coplien, Neil B.Harrison, 組織パターン, 翔泳社, 2013
[4] Mitsuyoshi Kozaki, Yoshifumi Kosumi, Tatsuhiko Terada, Naomi Honda , Toward High-quality Agile Software Development: NEC's Agile Development Management Method, ProMAC 2013.
[5] Norman L. Kerth, Dorset House, Project Retrospectives: A handbook for Team Reviews, 2001
[6] ケン・シュエイバー, マイク・ビードル, アジャイルソフトウェア開発スクラム, ピアソン・エデュケーション, 2003
[7] 天野勝, プロジェクトファシリテーション 実践編 ふりかえりガイド, http://objectclub.jp/download/files/pf/RetrospectiveMeetingGuide.pdf,
[8] Ken Scwaber, Jeff Sutherland, スクラムガイド, http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-JA.pdf