2004－SE－145 (3)
2004／8／19

普通のプロジェクトへの適用を目指したアジャイルな開発手法の構築と適用結果

藤井 拓[†], 鶴原谷 雅幸[†], 大津 尚史[††]

本論文では，平均的なスキルを持ち，アジャイルではない既存の開発手法に慣れた開発者及びプロジェクト管理者に受け入れられるアジャイルな開発手法 AMOP を構築するとともに，その適用結果を報告する．AMOP は，要求変化に対応する柔軟性とソフトウェアの品質向上を両立させるためにテストの自動化を大きな柱とし，アジャイルな開発手法であるXP及びスクラムのプラクティスを参考に定義された．AMOP を2つのプロジェクトに適用した結果，目標管理を始めとする各プラクティスが概ね良好に実践できており，　3ヶ月程度の期間内に期待通りあるいはそれを上回る機能を提供するプログラムが開発された．

Construction of an Agile Methodology for Ordinary Projects and Its Application Results

Taku Fujii[†], Masayuki Tsuruharaya[†], Hisashi Otsu[††]

Construction of an agile method aimed for ordinary projects, named AMOP, and its application results to 2 projects are reported in this paper. AMOP is designed to emphasize test automation to achieve both resilience of changes and better software quality, and its practices are defined by referencing practices of other agile methodologies; XP and Scrum. AMOP is applied to two projects in which most of practices including target management work well and successfully released equal or more than expected functions in about 3 months.

1. はじめに

Agile とは，英語で「機敏な」という意味の言葉であり，Agile なソフトウェア開発とは，機敏にソフトウェア開発を行うことを意味する．本論文では，以降 Agile を「アジャイル」とカタカナ表記することにする．近年日本で話題になっている XP(eXtreme Programming)[1]は，このようなアジャイルな開発の一つの具体例である．アジャイルなソフトウェア開発手法としては，XP だけに留まらずスクラム[2]，適応型ソフトウェア開発[3]などの複数の手法が提案されている．

2001 年 2 月に XP，スクラムなどの提案者から成る 17 名の開発方法論者が集って，自分達が目指すアジャイルなソフトウェア開発の価値を以下のようなアジャイル宣言(Agile Manifesto)[4]としてまとめた．

- プロセスやツールよりも個人や相互作用
- 分かりやすいドキュメントよりも動くソフトウェア
- 契約上の駆け引きよりも顧客とのコラボレーション
- 計画を硬直的に守るよりも変化に対応する

同時に，この宣言を行った方法論者によりアジャイルなソフトウェア開発の概念を普及，発展させるための NPO であるアジャイルアライアンス(Agile Alliance)[5]が設立された．

これらのアジャイルな開発手法は，開発プロセスの観点では要求，設計，実装，テストのサイクルを重ねながら開発を進める反復型開発アプローチ[6]の1種と見なすことができる．本論文では，要求，設計，実装，テストなどを通じて動作するソフトウェアを作成する1回のサイクルを反復(iteration)と呼ぶ．

アジャイルな開発手法は価値や原則やプラクティスで開発手法を定義しており，作業や成果物や役割で定義する伝統的な開発プロセスの方法とは異なる．このような価値や原則やプラクティスによる開発方法論の定義の狙いと期待される効果を示したものが表 1である．

† (株)オージス総研技術部ソフトウェア工学センター
†† (株)オージス総研ソリューション本部 OTS 第 3 部

表 1 価値や原則やプラクティスで開発手法を規定する大きな狙いと期待される効果

大きな狙い	期待される効果
開発者の自律性を尊重する	従来のプロセスのように機械的に決められた作業を決められた作業順序で実行するのではなく，少数の指針だけを定めることにより開発者が自律的に行動することで，要求内容や技術の変化に柔軟に対応することが可能になる
開発者が行うべきことを絞る	開発者が行うべきことを比較的少数のプラクティスに絞ることで開発作業の効率を向上し，ソフトウェアを作るという最も大事な任務に開発者が集中することを可能にする
官僚主義を排除する	作業の種類や範囲を細かく分割しないことにより，開発者毎の縄張り意識を減らし，状況に応じて開発者が互いに補える組織を作る

また，アジャイルな開発手法では開発者が相互の作業状況や知識を共有し，相互に助け合う組織を作り，ドキュメント作成の負荷を削減するために開発者間のコミュニケーションを促進することの重要性が強調されている．

その一方で，アジャイルアライアンスが提案する価値や原則は，抽象的なレベルに留まっているため，実際に開発作業をどのように進めるべきかイメージするのは非常に困難である．また，開発依頼者の要求及び技術の変化に柔軟に対応するというアジャイルアライアンスが掲げた期待効果が，開発者の相互作用を高める等々の価値や原則だけで実現できるのかどうかという疑問も残る．

本論文では，既存の開発手法に慣れた平均的なスキルの開発者でも受け入れられるようなアジャイルな開発手法を構築し，実践した結果を報告する．本研究で構築したアジャイルな開発手法を AMOP (Agile Methodology for Ordinary Projects)と呼ぶ．AMOP では，要求変化に対応する柔軟性とソフトウェアの品質向上を両立することを狙って，テストの自動化を大きな柱としている．

本論文では，まずアジャイル開発手法の代表例である XP の長所及び平均的なプロジェクトへの適用を困難にする点について論じ，その課題を AMOP でどのように克服しようとしているか説明する．つぎに，AMOP での開発作業の指針となるプラクティス及びプラクティスの選択において考慮した点を説明する．さらに，AMOP の実際のプロジェクトへの適用結果を報告する．

2. XP の長所と課題

XP は，アジャイルアライアンスの価値や原則を補完する複数のプラクティスを提案している．それらのプラクティスの特徴は，以下のとおりである．

- 設計，実装，テストなどの具体的な作業を実践するための指針となるプラクティスを提供している
- “テスト駆動型開発”や”ペアプログラミング”などの品質向上を狙ったプラクティスを加えることで，要求変化に対応する柔軟性とソフトウェアの品質向上を両立させようとしている

ここで，“テスト駆動型開発”とはテストコードをソフトウェア本体の実装コードに先行して実装する開発方式であり，“ペアプログラミング”とは2名の開発者がペアを形成し，1台のコンピュータ，キーボード，マウスを共有して開発を進める開発形式である．さらに，“テスト駆動型開発”を実践する際に単体テストの自動化を支援する JUnit などのツールの利用を推奨している．

その一方で，XP が推奨する“テスト駆動型開発”や“ペアプログラミング”に対しては，既存の手法に慣れた開発者や管理職には以下の2点で大きな抵抗がある．

- 製品ではなく，テストコードから先に考えることが求められ発想の転換が必要になる
- 労力に見合った効果が一般的に得られるかどうかの確証がない

表 2 XPとスクラムの比較

手法名	魅力的な点	心配な点
XP	● プロジェクト管理，設計，テスト，実装など作業に直接結びつく具体的なプラクティスを提供	● 開発途上で発生する問題の解決メカニズムが不明確 ● ペアプログラミング以外のコミュニケーション促進メカニズムが不明確
スクラム	● コミュニケーション促進メカニズムが明確 ● 開発途上で発生する問題の解決メカニズムが明確	● 設計，テスト，実装など作業に直接結びつく具体的なプラクティスがない

これら2点について抵抗があることが，XPが広範なプロジェクトに普及するための阻害要因となっていると著者らは考えた．

その一方で，要求変化に対応する柔軟性とソフトウェアの品質向上を両立しようというXPの考え方は，アジャイル開発手法の付加価値となりうる重要なポイントである．著者らは，そのポイントをテストの自動化により実現し，かつ既存の手法に慣れた開発者や管理職に受け入れられるアジャイル開発手法を構築しようと考えた．

3. AMOPの構築

著者らは複数のアジャイルな開発手法のプラクティスや作業を調べ，既存の開発手法に慣れた平均的なスキルの開発者でも受け入れられるようなプラクティスや作業を抽出し，アジャイルな開発手法を構築することを試みた．

3.1. アジャイルな開発手法の選定

AMOPの策定にあたり，まずAMOPを構築するためのプラクティス抽出源となるアジャイルな開発手法を選定した．選定にあたりXP，スクラムを始めとするいくつかのアジャイルな手法を調査したが，以下の3点を重視して，選定を行った．

A) 開発作業と直接結びつく具体的なプラクティスがあるか
B) コミュニケーションを促進する方法が明確かつ簡単かどうか
C) 開発途上で発生する問題の解決を促進する方法が明確かどうか

これら3つのポイントは，作業の指針を具体化する(A)，既存の手法との差別化を行う(B,C)ことを目指して設定された．

これらの3点を考慮した結果，表2に示されるようにA)のポイントにもっとも適合する手法がXPであり，B),C)のポイントにもっとも適合する手法がスクラムだと評価した．そこで，これら3つのポイントに適合するXPとスクラムをAMOPのプラクティス抽出源として用いることにした．

3.2. プラクティスの取捨選択

3.2.1. XPから取り入れた点

XPの大きな特徴の一つは，2.で述べたように“テスト駆動型開発”や”ペアプログラミング”のプラクティスを実践することにより，要求変化に対応する柔軟性とソフトウェアの品質向上を両立させる点にある．しかし，前述したように“テスト駆動型開発”や”ペアプログラミング” は既存の手法に慣れた開発者や管理職に受け入れがたいという問題点がある．

そのため，本研究では，要求変化に対応する柔軟性とソフトウェアの品質向上を現実的に両立するために，以下のような方針を適用することにした．

- ”ペアプログラミング”は実践しない
- “テスト駆動型開発“は，”反復で実装したコードに対して可能な限り自動化された単体テストを実装する“という形に大幅に緩める

同様に，XPの各プラクティスについて効果の確信がなかった”コーディング標準”及び指導なしの実践が困難だと思われる” 設計改善”を外した．また，“顧客テスト”，“シンプルな設計”，“メタファ”については必須とせず，適当と思われる局面でのみ限定的に実践すること

表 3 XP のプラクティスの取捨選択結果

○: 採用, △: 部分的に採用, ×:不採用

プラクティス	選択結果
全員同席	○
計画ゲーム	○
顧客テスト	△
短期リリース	○
シンプルな設計	△
ペアプログラミング	×
テスト駆動型開発	△
設計改善	×
共同所有	○
コーディング標準	×
常時結合	○
メタファ	△
適切なペース	○

にした.

XP のプラクティスを取捨選択した結果を表 3に示す.

3.2.2. **スクラムから取り入れた点**

スクラムではプラクティスを明示的に定義しておらず, 以下のような役割及び開発の進め方を提案している.

A) プロダクトの責任者が開発対象となる大きなレベルの機能のリストを"プロダクトバックログ"として書き出す

B) 開発は 30 日間のスプリントと呼ばれる反復を単位に進行する

C) プロダクトの責任者は各反復で実現すべき"プロダクトバックログ"の項目が割り当て, 開発メンバーはその項目の詳細仕様と必要なタスクを抽出する

D) 開発メンバーは毎日決まった時刻に作業実績, 作業予定, 問題点を報告する"デイリースクラム"という打ち合わせを開催する

E) 開発途上で発生した問題点を解決し, プロジェクトが順調に開発を進めることに責任を持つ"スクラムマスター"と呼ばれる役割を設定する

これら5項目の中で, A), D), E)は各々XP のプラクティスを大局的な開発範囲の設定, コミュニケーションの促進と問題点の顕在化, 問題点の解決の促進という点で補う項目だと考えられるので, AMOP に取り入れることにした.

一方で, 以下の2点を意図して AMOP では B)は採用せず反復の期間は2週間に設定した.

- 開発依頼者から開発内容に対するフィードバックをなるべく早く得る
- 反復の計画と実績を対比するサイクルを短くすることにより, 開発者の計画能力を早期に育成する

また, C)については XP の計画ゲームとかなり類似性が高いが, 計画に関する作業内容がより大雑把に記述されている. そこで, AMOP では作業内容がより詳しく記述されている XP の計画ゲームの計画作業のステップを取り入れた.

表 4 AMOP のプラクティス

プラクティス	説明
2週間の反復サイクル	2週間単位の目標管理及び製品のフィードバック
テストコードの作成	反復内で実装したコードに対して自動実行できるテストコードを可能な限り実装
計画ゲーム	ユーザストーリとタスクに基づく目標設定
デイリースクラムとスクラムマスター	プロジェクトの障害発生を常にモニターし, 障害による遅延を極力防止
常時結合と共同所有	コードは逐次構成管理システムに登録し, 日々ビルド及びテストを実行
適切なペース	メリハリの無く長時間労働をしない

表 5 AMOPを適用した2つのプロジェクトの概要

		A	B
開発内容		社内向け開発ツール	UML モデリングツールの拡張機能
開発期間		4ヶ月間	3ヶ月間
予備検討期間		1ヶ月間	1週間
開発体制	製品責任者	1(兼務)	1(兼務)
	スクラムマスター	1	1
	メンバー	1	2
必要となる要素技術や製品		● J2EE ● J2SE ● OpenOffice Writer	● J2SE ● UML モデリングツール
テスト自動化ツール		● JUnit ● JUnit for Struts	● JUnit

表 6 プロジェクトメンバーの開発及び技術経験

プロジェクト	役割	開発経験(年)	Java言語経験(年)	要素技術や製品の経験(カバー率)
A	スクラムマスター	10	0.4	2/3
	メンバー	0	0	0/3
B	スクラムマスター	0.25	0.25	1/2
	メンバー	10	0.5	1/2
	メンバー	6	6	0/2

以上説明したようなXPとスクラムのプラクティスの取捨選択の結果から作成されたのが表 4に示されるAMOPのプラクティスである.

4. AMOPの実践結果

現在までのところ, AMOPを2プロジェクトに適用した. これらの2プロジェクトを便宜的に各々Aプロジェクト, Bプロジェクトと呼ぶ. これら2つのプロジェクトの概要を表 5に示し, プロジェクトメンバーの開発及び技術経験を表 6に示す. A,B 両方のプロジェクトのスクラムマスターは, 問題解決だけを行うだけではなく, 製品の設計, 実装, テストの作業も担当した. B プロジェクトのスクラムマスターは, B プロジェクト以前にプロジェクト管理の経験はなかったが, A プロジェクトの開発メンバーとして計画ゲームの実践を経験していた. また, A プロジェクトではスクラムマスター, 開発メンバーともに社員だったが, B プロジェクトではスクラムマスターが社員, 開発メンバー2名はスクラムマスターが過去共に仕事をした経験がない協力会社の要員だった.

両プロジェクトとも開発初期にプロジェクトで使用する技術の調査や技術修得のための予備検討期間を置き, その後2週間単位の反復により開発を進めた. 各反復の初日では, 以下のように計画ゲームにより開発目標を設定し, 反復中に進捗をモニターした.

① 今反復で実現すべき機能をユーザストーリという形で書き出す
② プロダクトの責任者が各ユーザストーリの優先度を設定するとともに, スクラムマスターが労力を見積もる
③ 2週間という反復の期間内で実現可能なユーザストーリをプロダクトの責任者が選択する
④ 選択されたユーザストーリを各メンバーに割り当て, 必要なタスクのリストを定義し, ②で見積もった労力が妥当かどうかチェックする
⑤ ④で労力の見積もりの不整合が発生した場合, プロダクトの責任者とともに再度目標となるユーザストーリの範囲を見直す
⑥ 最終的に確定したユーザストーリとタスクは, SWiki[7]という協調作業用のツールに記入される
⑦ 反復途上で各タスクやユーザストーリが完了したら, 完了日付をSwiki上に記入する

図 1は, Aプロジェクトにおいて反復毎のユーザストーリ数, 反復最初で設定されたタスク数及び実際に完了したタスク数を示したものである.

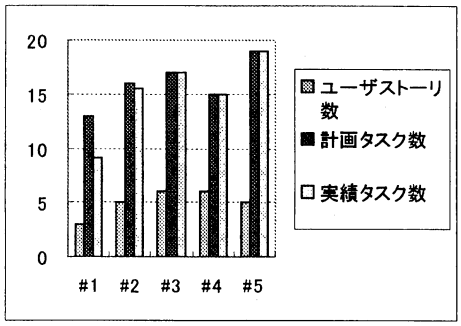

図 1 ユーザストーリ，タスクの計画と実績
(A プロジェクト)

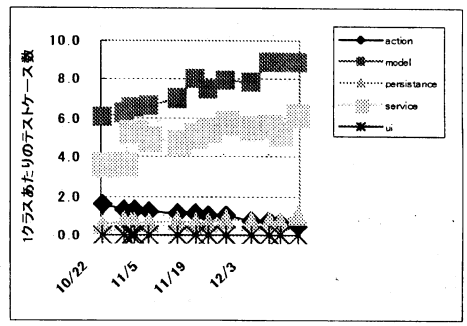

図 2 1クラスあたりのテストケース数の推移
(A プロジェクト)

この図から，第2反復以降計画したタスクはほぼ完全に消化できたことが分かる．

図 2は，A プロジェクトの主なパッケージ毎にアプリケーションクラス1クラス当りの平均テストケースを求めた結果を示す．この図のテストケース数は，JUnit 等で自動実行できるテストケースクラスの testXX 操作数に対応する．また，ui パッケージのクラスについては，A プロジェクトで用いた JUnit などのツールにより単体テストが自動化できなかったため，テストの実行状況はモニターできていない．

この結果より，開発途上でもテストケースが順次拡充されていることが分かる．

A プロジェクトの終了時点で，テストの網羅性を確認するために単体テストが自動化されたパッケージ中の3パッケージのラインカバレージを計測した．その計測結果を表 7に示す．action パッケージについては，単体テストが自動化されたものの，カバレージ計測ツールの不具合で正常な計測値が得られなかったので除外してある．

この計測結果より model と persistence パッケージについては，比較的良好なラインカバレージが達成されていることが分かる．その一方で，service パッケージのラインカバレージは不十分であることが判明した．

このように service パッケージのラインカバレージが低い値に留まったのは，A プロジェクトの開発途上でカバレージ計測を行わず，テストケースが十分であるかどうかという判断を開発者の主観に委ねたことが主原因だと考えられる．

表 7 プログラムコードのラインカバレージとコード行数 (A プロジェクト)

パッケージ名	ラインカバレージ(%)	コード行数
model	94	572
persistence	73	376
service	40	1324

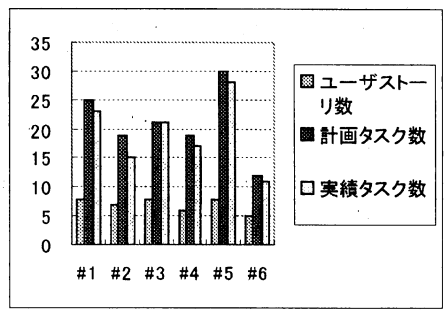

図 3 ユーザストーリ，タスクの計画と実績
(B プロジェクト)

図 3は，B プロジェクトのユーザストーリ及び計画したタスクの消化状況を示す．B プロジェクトでは第1反復は製品の細かい仕様策定に当て，実際にプログラムが作られたのは第 2 反復からだった．また，第 6 反復は開発メンバーの契約期間の制約で1週間と他の

反復の半分の期間に設定した.

このような結果より, B プロジェクトにおいても目標管理はうまく機能したことが分かる.

一方, テストの面では B プロジェクトの開発対象はGUIアプリケーションであったことやJUnitのような単体テストツールでテストの初期状態の制御を簡単に行えなかったため, 開発期間内でのテストの自動化はあまりなされなかった. また, Bプロジェクトの途中でGUIテストツールの導入によるテストの自動化を試みたが, いくつかの問題に直面し, 適用を断念した.

B プロジェクトの完了後, この GUI テストツールの問題が解決したため, GUI テストツールでテストスクリプトを生成, 実行し, ラインカバレージの計測を行った. その結果, 3日間程度で主要な機能を97%程度のラインカバレージで網羅するテストスクリプトを作成できた. この追加検討から, GUI テストツールが開発初期に導入されていれば, B プロジェクトでも開発途上のテストの自動化が達成できたと考えらえる. さらに, プロジェクト完了後 JUnit のような単体テストツールで本当にテストの初期状態の制御ができないかという点について詳しく調べたところ, ある程度の工夫を行えば初期状態の制御ができたはずであることも判明した. これらの検討からテストの自動化については, プログラミングが本格化する予備検討期間においてテストについて詳しい人間の支援も受けて具体的な実現方法を決めることが必要であることが分かった.

A,B 両方のプロジェクト共に目標管理が良好だった点については, 筆者らは以下の2点が大きかったと考えている.

A) スクラムマスターがプログラムコードやテストコードを理解できるため, コミュニケーションや問題解決が円滑に行えた
B) 問題解決において, スクラムマスターが自分自身で考えるだけではなく, 開発メンバーの意見やインターネット上の情報をうまく取り入れた

逆に, スクラムから派生したAMOPのような開発手法を用いる場合には, これらの2点をある程度備え, 心がけることでプロジェクト管理経験がない若手エンジニアでもプロジェクト管理が可能になると考えられる.

5. まとめ

本論文では, 平均的なスキルを持ち, アジャイルではない既存の開発手法に慣れた開発者及びプロジェクト管理者に受け入れられるアジャイルな開発手法AMOPを構築し, その適用結果を報告した.

AMOP は, 要求変化に対応する柔軟性とソフトウェアの品質向上を両立させるためにテストの自動化を大きな柱としている. また, アジャイル宣言の価値や原則を補完する作業の具体的な指針を定義するためにXP及びスクラムのプラクティスを参考にしながらAMOPのプラクティスを定義した.

現在までのところ, AMOP を2つのプロジェクトに適用し, 目標管理に関するプラクティスは概ね期待通り実践できており, テストの自動化については事前準備を行えば達成可能だという結果が得られた. また, テストの網羅性を客観的に評価するためにテストカバレージの自動計測を行うことが重要だということが分かった. また, プロジェクト管理においてはスクラムマスターという問題解決を主任務とする役割を設定することで, プロジェクト管理経験がない若手エンジニアでもプロジェクト管理が可能になるという可能性が示された.

現在の AMOP は, 開発者がモデリングスキルを持っていないとの仮定で方法論を考えたため, これまでモデリングを積極的に取り入れてこなかった. しかし, 開発依頼者との間で要求を議論したり, 開発者の間での設計情報を議論及び共有するためにはモデリングが有効だと考えられる. 今後, AMOP にモデリングをどのように取り込むかを考えることが大きな課題となる.

参考文献

[1] ケント・ベック, *XP エクストリーム・プログラミング*, ピアソン・エデュケーション, 2000
[2] ケン・シュエイバー他, *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
[3] ジム・ハイスミス, *適応型ソフトウエア開発 – 変化とスピードに挑むプロジェクトマネージメント*, 翔泳社, 2003
[4] http://www.agilemanifesto.org/
[5] http://www.agilealliance.org/home/
[6] ウォーカー・ロイス, *ソフトウェアプロジェクト管理 – 21世紀に向けた統一アプローチ*, ピアソン・エデュケーション, 2001
[7] ボウ・ルーフ他, Wiki Way—コラボレーションツール Wiki, ソフトバンクパブリッシング, 2002