社内システム開発アジャイル化における技術課題の解決とチーム構築の経緯

アドバンストテクノロジー本部
清水　弘典

アドバンストテクノロジー本部
ビジネステクノロジー部
安藤　徹也

1. はじめに

1.1 本稿の対象範囲

17名のソフトウェア開発者によりアジャイルソフトウェア開発宣言が文書化されたのが2001年である。書籍「XPエクストリームプログラミング入門」の日本語版は初版が2000年であるから、アジャイルはその誕生とほぼ同時期に日本でも紹介されていたことになる。当時はXPが主に紹介されていたこともあり、物珍しさが先行していたと記憶している。その後、およそ20年の間にPMBOKにもアジャイルが取り込まれるなど、一定の支持を得ている感がある。

ただし、日本における受託開発プロジェクトは今でもウォーターフォール型が主流である。IPAの「ソフトウェア開発データ白書2018−2019」によると、データ収集したプロジェクトのうち、プロジェクトの形態は受託開発が89.3%、開発の進め方はウォーターフォール型が97.4%となっている。これらの数値から推測すると、アジャイルに染まっている人材はまだ少ない様子である。

本稿で想定している読者は全身をアジャイルに染めている人ではない。アジャイルに染まっていないが、興味を持っている人を対象にしている。たとえば、アジャイルでプロジェクトを立ち上げたら実態はどんな状態になりがちなのか興味がある人、または草の根的にアジャイルのプラクティス（チケット管理や単体テストの自動化など）を部分的に試していた人、そんな読者を念頭に置いている。

本稿を通じて、自称「草の根アジャイラー」が「フルスペックなアジャイル」を5ヵ月間、観察してきたなかで感じた違和感やその改善を探る方法をお伝えできたら幸いである。

1.2 本稿の構成

第2章は開発現場の動向と期待されていることを整理する。第3章はそれに対する当社の取り組みを紹介する。第4章は取り組みの中で実践している技術面について掘り下げて紹介する。第5章はチーム作りや人材面について掘り下げて紹介する。

各章は独立しているので、技術面に興味がある方は第4章だけ、チーム作りや人材面に興味がある方は第5章だけでも目を通していただけたら幸いである。

2. 開発現場の動向と期待

2.1 アジャイルへの期待

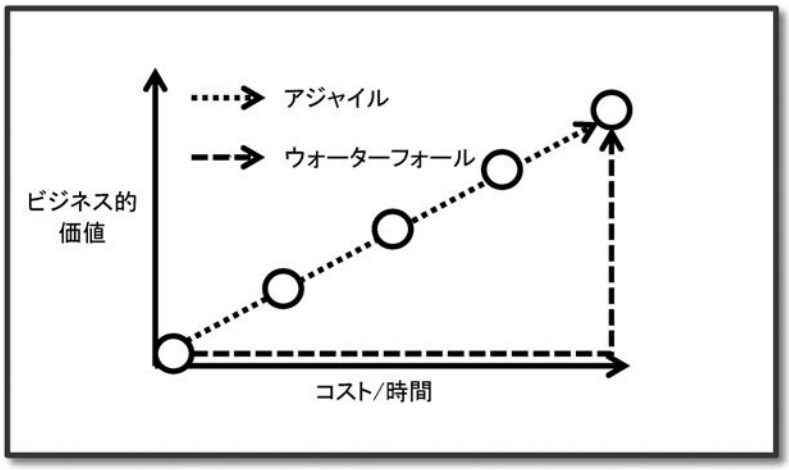
アジャイルはQCDのうちDelivery（デリバリー）を重視するシステムの期待に応えられる開発手法である。エンドユーザーは要望を出してから数週間（早ければ数日）で、欲しい機能を使えるようになる。その代わり、Quality（品質）は80点で良い（後から改善すればよい）という割り切りも必要である。

近年のDX領域のようにビジネスの環境変化へ素早い対応を重視する分野においては、リリースまでのリードタイムが短い開発手法が求められている。また、AIのような確立していない分野を含んでいる場合、ビジネスの先鞭をつけたいからといって、システムに一括して多額の投資を行うのは危険を伴う。

早くリリースしたい一方で投資は抑えたい場合や、仮説と検証を繰り返しながら進めたい分野において、アジャイルの逐次性はウォーターフォールの一括性よりも向いている。

つまり、必要な機能をすべてではなく少しずつで良いからタイムリーに利用したい分野からの期待に対して、アジャイルは第一選択肢となっている（図2.1）。

図2.1　アジャイルの逐次性



2.2 クラウドへの期待

いざアジャイルを始めようとすると、意外と環境構築に手間がかかる。アジャイルを推進するには多くのサーバーが必要で、ハードウェア調達やセットアップに知識と時間を必要とする（ソース管理用、ビルド兼単体テスト用、QA用、本番用、チケット管理用、ドキュメント保存用など）。

かつて、ハードウェア調達はアジャイルの有無にかかわらずリードタイムを数週間程度は確保しておく必要があった。ガントチャートを作成する際、クリティカルパスとして頭を悩ませたものである。

いまは、クラウドの普及で、まずハードウェア調達によるクリティカルパスがなくなった。またAzure DevOpsのようなクラウドサービスを使うことで、アジャイルに必要なサーバーの準備が容易になってきている。

つまり、短期間でアジャイルを始めたい開発チームからの期待に対して、クラウドは第一選択肢となっている。

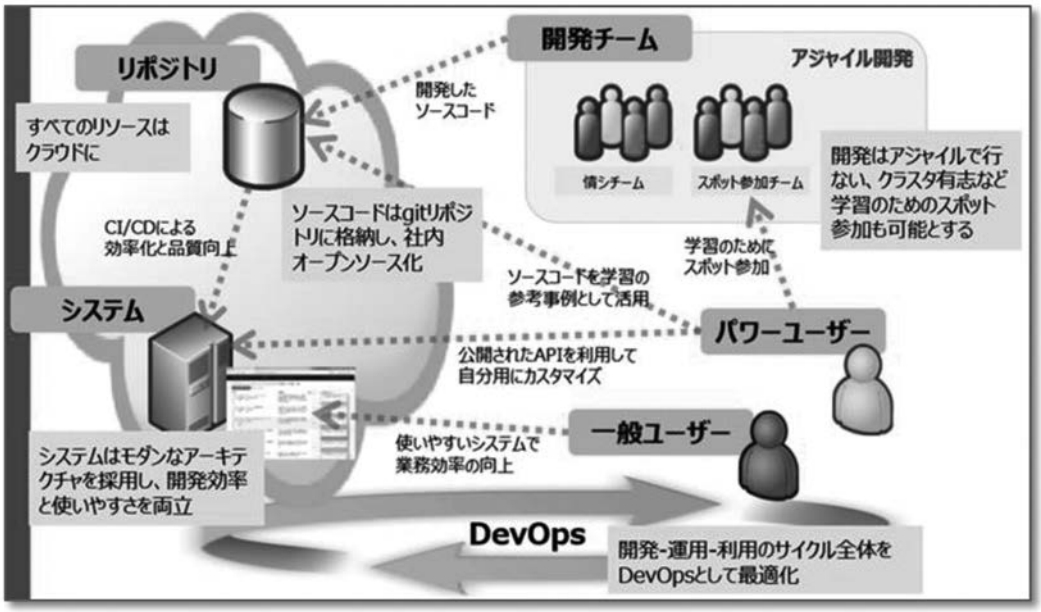
3. 当社の取り組み

3.1 実践プロジェクトの企画

当社では、社内システムの更改を経験の少ない分野に対するチャレンジ、知見の蓄積、技術者育成の場として活用している（図3.1）。

現在、パイロットプロジェクトを立ち上げ、必要な体制、プロセス、ツール等の実践検証を行っている。なお、パイロットプロジェクトについては第4章で紹介する。

図3.1　当社の取り組み



4. 技術課題と解決について

4.1 技術課題

4.1.1 プロジェクト概要

パイロットプロジェクトとして、当社の社内システムの1つである要員調達システム「KaNAME」の開発をしている。プロジェクトの開始時点では開発メンバーは4名であった。開発の単位であるスプリントは当プロジェクトでは基本2週間としている。2週間の中で計画・設計・プログラミング・テスト・レビュー・リリースを行う。

4.1.2 開発環境

プログラム言語はC#、Webアプリケーションフレームワークは ASP.NET Core MVC、DBアクセスフレームワークはEntity Framework、フロントエンドのライブラリはBootstrapを使用している。DBはSQL Server、バージョン管理はGit、統合開発環境はVisual Studio2019を使用している。テスト環境は開発メンバーが使用する統合環境と、プロダクトオーナーが確認に使用するステージング環境の2つをAzure上に用意している。

4.1.3 アーキテクチャ

アーキテクチャとしてはMicrosoft Azureのクラウドサービスを主に利用している。Webホスティングサービスとしては Azure Web Apps、DBはAzure SQL Databaseを利用している。認証はAzure Active Directory、ログ監視はAzure Application Insightsを利用している。メール機能についてはAzureに該当するサービスが無いため、クラウドベースのメール配信サービスであるSendGridを利用している。

開発メンバー内にC＃やAzureに詳しい人がいない中での開発となり、不明点については社内の有識者に質問したり実装に入る前に動画学習サービスのUdemyのコースを利用して対応を行っている。

4.1.4 Git

プログラムのバージョン管理はGitで行い、Gitクライアントとしては Visual Studio2019とSourcetreeを使用している。Gitに慣れているメンバーがいない中での利用となり、開発当初はGitの操作とローカルとリモートの2つにリポジトリを持つという考え方に苦労をした。ブランチ管理のルールであるブランチモデルとして何を採用するか迷ったが「git-flow」を採用している。理由としては本番環境・ステージング環境・統合環境の各テスト環境とブランチが一致しており、修正時にどのブランチを直せばよいかイメージしやすいことと Sourcetreeがgit-flowに対応していることからである。当チームではブランチとしてmaster、develop、feature、release、hotfixの5種類のブランチを定義している。featureブランチについて当初は開発者ごとにブランチを作成していたが、開発者が担当するユーザーストーリー毎にブランチを作成することにしている。

Gitを使用するメリットは、本番環境と開発環境の各環境の断面の資源を取得しやすく修正に入りやすいことと、お試しで修正して確認することがしやすいことである。Azure DevOpsで利用できるプルリクエストの機能を用いてコードレビューできることも便利である。ただしGitの初心者がGitの考え方と操作に慣れるのにある程度時間がかかっている。

4.1.5 CI/CD

早いペースでリリースを行うためにテスト・ビルド・リリースを自動化する必要があり、これらを自動化するための手法が「CI/CD」である。当チームではAzure DevOpsのPipelinesの機能を利用してCI/CDを導入している。Pipelinesの Builds機能でビルドと単体テストの自動化を実施し、Releases機能で各環境へのリリースを実施している。

CI/CDのメリットとしては、テスト環境での確認がすぐできることが挙げられる。また、リリース漏れが発生しにくいということもメリットとして挙げられる。

4.1.6 自動テスト

自動単体テストツールとしてMSTestを使用して単体テストの自動化を導入している。ローカルでの開発時に単体テストを動かし、Azure DevOpsでビルドをする度にも単体テストを動かすようにしている。テストコードで使用するモックライブラリとしてMoqを利用している。

苦労したこととしてはDBアクセスの処理を自動テスト化することがある。DBに対するデータアクセスをモックオブジェクトを通じて疑似的に行えるよう、プログラムをリポジトリパターンに書き直すことで対応した。

自動テストのメリットとしては、一度ソースをかけば自動でテストできることが挙げられる。デメリットとしては、実装するたびにテストコードを追加・修正する必要があることが挙げられる。

当チームの自動テストはデグレードが発生していないことを確認するためのリグレッションテストの意味合いが強い。仕様があっているかどうかのテストはローカル環境での画面打鍵テストで実施している。

4.2 プラクティス

4.2.1 コミュニケーション計画

開発チームは目印となる旗を立てて同じテーブルで作業をしている。メールよりもMicrosoftのTeamsの会話とチャットの機能を多く使っている。タスク管理はMicrosoftのAzure DevOpsのBoardの機能を利用している。残作業量を視覚化する「バーンダウンチャート」もBoardから確認でき活用している。

日々の進捗確認として開発チームで毎日「DailyMeeting」を実施している。15分程度の時間でそれぞれが昨日実施したタスクと今日実施する予定のタスクを発表している。当日のタスクの割り当てがない人については、タスクの割り当てを最後に行うことにしている。タスクを割り当てる方法は、未割当のタスクのうち優先度に従って各メンバーがやりたいタスクを割り当てるようにしている。

スプリントの最初には「スプリント計画」の打ち合わせを開発チーム、プロダクトオーナー、スクラムマスターで実施している。

スプリントの中間には「スプリント調整会議」を実施している。進捗の報告とスプリント終了までに完了するユーザーストーリーと完了しないユーザーストーリーの確認をプロダクトオーナーと行っている。

スプリントの終了時にはプロダクトオーナーと「スプリントレビュー」を実施している。

プロダクトオーナーと仕様の確認を行う、「プロダクトオーナーミーティング」をスプリント中に2回実施している。プロダクトオーナーと仕様の認識のずれを早く見つける目的で実施している。

1週間の終わりには、その週の中で良かったことと悪かったことを開発メンバー全員が発表する「ふりかえり」の打ち合わせを実施している（ふりかえりについては、4.2.4などで詳述する）。

4.2.2 スプリント計画

スプリントの初めにはスプリント計画を行っている。スプリント計画ではユーザーストーリーの概要・テスト方法・優先度を確認する。「ユーザーストーリー」とは、顧客がソフトウェアで実現したいと思っている機能や特徴を簡潔に記述したものである。

また、スプリント計画では見積もりを開発チーム全員で行う。アジャイルでは見積もりを「ストーリーポイント」と呼ばれる相対値で扱っている。開発チームで「プランニングポーカー」を実施することでストーリーポイントを求めている。プランニングポーカーは１、２、３、５、８とポイントが書かれたカードを各メンバーが用意することから始まる。基準となるユーザーストーリーを3ポイントと決めて、それと比べて他のユーザーストーリーが相対的に何ポイントになるか各メンバーがカードを同時に出すことで決めている。ポイントがメンバー内で一致すればそのポイントで決定になる。ポイントが食い違えばそのポイントを出した理由を各メンバーが発表して、再度カードを出し、ポイントが一致するまで続ける。

プランニングポーカーを実施してみての感想としては、カードを出して見積もりを決めていくことがゲーム感覚で楽しく、メンバーの意見を聞くのでユーザーストーリーについて開発メンバー間の認識の一致を図るのに役立っていると思う。

4.2.3 スプリントレビュー

スプリントの終了時にはスプリントレビューをプロダクトオーナーに対して行っている。対応したユーザーストーリーのデモンストレーションを開発チームが行いプロダクトオーナーから了承や指摘を受け取っている。チームの開発速度を表す「ベロシティ」をAzure DevOpsから算出しプロダクトオーナーに報告を行っている。

4.2.4 ふりかえり

アジャイル開発では定期的に開発チームメンバーでふりかえりを行う。

当チームでは1週間の終わりに開発チームで集まって1週間の中で良かったこと・問題だったこと・その解決策を話しあうふりかえりの作業を行っている。

書籍「モブプログラミング・ベストプラクティス」で紹介されている「6つの帽子思考法」をベースにしたオリジナルの方法でふりかえりを実施している。1回のふりかえりは30分程度で行っている。

「6つの帽子思考法」は視点を切り換えてアイデアを出すため、実際に実施してみるとアイデアを出しやすい印象である。今回のチームでは「6つの帽子思考法」にある6つの視点のうち、事実と数値への視点（白色）、肯定的な側面への視点（黄色）、警戒と注意を促すマイナス面への視点（黒色）、プロセス全体を構成する視点（青色）、創造性と新しい考え方への視点（緑色）の5つの視点でふりかえりを行っている。各視点の内容を紙に書いてホワイトボードに貼っていき、開発チームメンバーで話し合いを行っている。ふりかえりで出た改善内容をメモに貼りだして、次スプリントから改善するように努めている。改善内容は、いつ何をするかを具体的に決めて次スプリントで実行するようにしている。問題点がバラバラでまとまらないときには将来1番リスクとなるものを問題点として選んでいる。前述したプロダクトオーナーミーティングはふりかえりの結果から実施することになったものである。仕様がプロダクトオーナーの想定と違うユーザーストーリーが見つかり、そのことがきっかけでプロダクトオーナーミーティングを開催することにした。

4.3 今後の課題

「KaNAME」は本番リリースを迎え、すでにユーザーに使われるようになった。しかし開発と運用を素早く行うために乗り越えるべき課題がある。

4.3.1 結合テストの自動化

画面打鍵テストを自動化することができておらず、手動による画面打鍵を行っている。これまでに、プログラム修正時に影響範囲を十分に確認せずに修正して想定していない機能で問題が発生することがあった。画面打鍵テストの自動化を行

いビルド時に自動実行することで問題の検知を早めることができると考えられる。全テストケースを自動化することは現実的ではないので、代表的なテストケースの画面打鍵テストの自動化を今後行いたい。

4.3.2 テーブル定義変更の反映の自動化

テーブル定義変更時のリリース管理を現在はExcelで行っている状態なので、テーブル定義変更の自動化を行いたい。DBマイグレーションのツールを利用してCI/CDの中に組み込む対応を行いたい。

4.3.3 マスターデータの自動更新

当チームでは開発だけではなく運用も一緒に行っている。マスター系のテーブルデータの更新が定期的に必要であるが、現在は手動で更新を行っている。運用にかかる工数を少なくするためにデータ更新の自動化が必要である。

4.3.4 属人化防止

特定の人が特定のタスクを行う傾向があるので属人化を防ぎたい。Azure環境に関するタスクやリリースに関するタスクが特定の人に固まりがちになっている。対応策として、自分しか知らない内容の作業を実施したらwikiに書き、タスクの担当者を割り当てるときにメンバーが固定しないようにすることで属人化を防ごうとしている。また、実装に関してそのスプリントで新しく登場したトピックがあれば、開発メンバー全員で共有するための説明会をスプリントの終わりに開催することを最近始めている。

5. チーム構築について

5.1 チームの範囲

ここでのチームとはスクラムチームを指すこととする。チームはプロダクトオーナー、開発チーム、スクラムマスターで構成される。いわば、企画と開発と運用が1つの部屋で一緒に机を囲むイメージである。

5.2 初期のアジャイルのチーム像

アジャイルを推進していく中でストレスが溜まっていくことがある。それは「アジャイルのプラクティスを実践しても何かうまくいかない。」ということだ。先人の知恵を拝借しているのに上手くいかないのはなぜか。

アジャイルを始めたての頃は、できそうなプラクティスをとりあえず試してみようということになる。しかし、試してみた結果がこれで良いのか、確信が持てない状況に陥ることがある。

たとえば、初めてプランニングポーカーで見積もりをしたときのことである。アジャイルを紹介する書籍を片手に持ちながら試してみると、たしかにストーリーポイントは決まる。しかし、決まりはするのだが、経験則からくる見積もりと肌感覚が合わない。ありていに言うと見積もりが大きすぎるのだ。

理由は開発メンバーがバッファを見込んでいたからなのだが、実際にスプリントを終了するとバッファを使い切った状態であった。結果だけ見ると、開発メンバーの保守的な見積もりが功を奏したように見える。しかし、これで良いのだろうか。

プランニングポーカーはユーザーストーリーをもとにして見積もりを行う。ユーザーストーリーは要求と受入条件で構成される。いわば外部設計（基本設計）を簡易化したものである。内部設計（詳細設計）、実装、テストは開発メンバーに任される。

たとえば、仮に工数を5日と見積もったら、5日なりの作業を行う。5日と見積もったのだから5日で完了するような個人作業を無意識のうちに積み上げてしまうのである。つまり、個人のゴールを目指すのみで、工夫をしてチームに貢献しようとする行動特性が無い状態である。

プラクティスのまねだけをしてもこのような問題には対処できない。なぜならプラクティスは手段に過ぎないからである。

つまり、この先人の知恵を生かすも殺すもアジャイルに適した行動特性を備えているかどうかにかかっている。この章ではそのような行動特性を備えるチームを作るための仕組みを探る。

5.3 目指すアジャイルのチーム像

ウォーターフォールは、バトン（成果物）を引き継ぐリレー競争にたとえられる。決められた経路（計画）を無駄なく走り、ゴールをめざすイメージである。個々のメンバーは事前に決められた自分の担当範囲においてベストを尽くす。また、組織構造はプロジェクトマネジャーを頂点とするピラミッド型で、メンバーには指示に従い（他律的）個人としての成果が期待されている。

一方、アジャイル（スクラム）はラグビーにたとえられる。チームが一群となって移動するかのように走り、ボール（成果物）をあちこちパスし、障害を柔軟にかわしながらトライ（ゴール）を目指すイメージである。個々のメンバーは、つねに変化するゲームのうごき全体を把握しながらチームにベストな行動を自分で選択する。

組織構造はフラット型で、メンバーには指示待ちではなく（自律的）、チームとしての成果に貢献することが期待されている。

まとめると、速さと柔軟性を重視するアジャイルにおいて、必要なのは他律的なピラミッド型ではなく、自律的なフラット型の

チームである。加えて、チーム内では他人の仕事に注意をはらい、個人のゴールではなくチームとしてのゴールに協力する関係ができており、自分の進む方向を自分で決めている。目指すのはそんなチーム像である。

考えたくもないが、仮に自律的でないアジャイルチームにアサインされたとしたならば、出口がみえない混沌の中をさまようことになるだろう。

5.4 チームビルディング

ここでは前述の目指すべきアジャイルチームを作り、成長させていくことをチームビルディングと呼ぶ。最初にメンバーとチームの成長に必要な課題を提示し、つぎに課題解決の仕組みづくりを紹介する。

5.4.1 メンバーの成長に向けた課題

仮に、足の速いリレー選手を集めてラグビーチームを作ったら、勝てるだろうか。足が速いのだから容易にトライ(ゴール)を狙えるようにも思える。しかし、おそらくラグビー選手のチームにはかなわないだろう。

同じことは開発チーム作りにも言える。優秀なウォーターフォール要員を集めてチームを作っても、自律的なアジャイルのチームにはならない。なぜならば、構成メンバーに求められるスキルが違うからである。

実際問題として、ウォーターフォールの経験が長いメンバーに、いきなり自律的に動くことを求めるのは酷である。とくにプロジェクトマネジャーからタスクを割り振られるスタイルの仕事(他律的な仕事)を続けてきたメンバーの場合、全てのタスクを見渡してチームに求められていることを理解し、自分が貢献できることを意思表示するスタイルの仕事(自律的な仕事)を身に着けるのは一朝一夕にできることではない。

したがって、他律的から自律的へとスキル転換すべく、メンバーを育てていく仕組みづくりが課題として浮上する。

5.4.2 チームの成長に向けた課題

ここに、プロジェクトマネジャーを頂点とするピラミッド型の開発チームがあるとする。プロジェクトマネジャーが成長するにつれて、チームも成長(QCDの達成度が上がるなど)していくだろう。しかし、その知見がプロジェクトマネジャーの属人化した暗黙知のままだとしたら、会社組織には損失である。なぜならば、形式知として共有する仕組みがないと、その知見が組織に蓄積・伝播し辛いからである。

プロジェクトマネジャーをアジャイルチームに置き換えても同じことが言える。アジャイルの知見が属人化ならぬ属チーム化してしまったままでは、プロジェクト終了でチームが解散すると同時に知見が消えてしまう。当然、その知見は組織に蓄積・伝播し辛いままである。

したがって、その知見を暗黙知から形式知として共有していく仕組みづくりが課題として浮上する。

5.4.3 課題解決に向けた仕組みづくり

前述の課題を解決する場として、当チームではふりかえりを活用している。具体的な方法は第4章に譲るが、大まかには、まず課題を設定し、つぎに改善策を議論し、改善案ができたら全員で合意する、という流れである。

実はこの流れのなかに3つの仕組みを忍び込ませている。

1つ目は、自律的なメンバーを育てるための仕組みである。具体的には、ふりかえりするときのガイドとして作成したパワーポイントのなかで、自分たちで課題を設定し、自分たちで改善を検討する流れにしている。他人から指示されるのではなく自分たちだけで継続的に一連の作業を繰り返す仕組みにより、他律的から自律的へスキル転換を促している。

2つ目は、チームのゴールを共有化する仕組みである。具体的には、課題の設定は個人の課題から出発してチームの課題を設定する流れにしている。個人のゴールだけでなく、チーム全体を見渡して共有のゴールに向かう行動特性をもつように促している。

3つ目は、チームの暗黙知を形式知として表出させる仕組みである。具体的には、ふりかえりの成果物である改善策がそれにあたる。ふりかえりの改善策には、もしふりかえりをしなかったならば、表出しなかった課題と改善策が明文化されている。

改善策はチームのルール集である。たとえるなら鉄道のレールを敷くようなもので、チームのルールを敷くことにより、プロジェクトの運行が速くスムースになっていく。

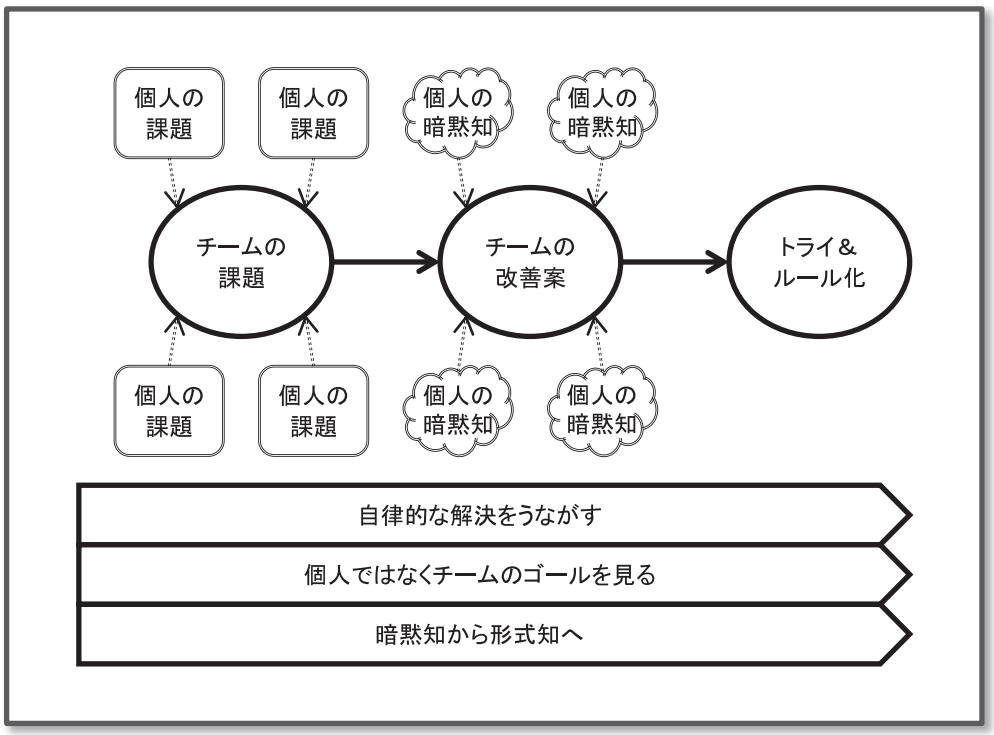
このような仕組みによって、目指すべきアジャイルチームへ成長していくことをねらっている(図5.1)。

5.5 今後の課題

現状のふりかえりには課題もある。チームが3～4人程度なら問題ないのだが、それ以上の人数になると、関わり方に温度差がでてくるのだ。そもそも個性が異なるメンバーをワンセットにしてふりかえりを行っても同じ認識を持つとは限らない。この課題については、1on1のような個性に合わせた仕組みづくりが別途必要ではないかと認識している。

別の課題として、チームのゴール共有がある。プロジェクトの性格として、メンバーの入れ替わりが多く発生する。メン

図5.1　課題解決の仕組みとねらい



バーが入れ替わっても素早く混乱期から機能期へと段階を移す仕組みづくりを模索中である。

アジャイルが広く知られる契機となったアジャイルソフトウェア開発宣言の提唱者達は豊富なアイデアやスキルを持った人たちであり、その人たちの視点でアジャイルは出発している。

毎回、それだけのスキルを持ったメンバーを集め、プロジェクトを立ち上げるのは非現実的である。それを補うのがスクラムマスターの存在なのかもしれないが、そのスクラムマスターをどう育成するか、これが一番の課題かもしれない。

6. おわりに

従来の開発手法が時代遅れで、すべてがアジャイルにかわっていく、ということはない。要は適材適所である。これらは手段であり目的ではない。従来の開発手法とアジャイルの長所短所を理解の上、使い分けて（またはミックスして）目的達成のために利用すべきである。

7. 謝辞

本稿を執筆するにあたり、複数の方にお話をうかがった。多忙の中、貴重な時間を割いて親身なアドバイスをしてくださった方々にこの場を借りて心からの感謝を申し上げたい。

参考文献

1）ケント・ベック:「XPエクストリーム・プログラミング入門−変化を受け入れる」、ピアソンエデュケーション、2005
2）独立行政法人情報処理推進機構（IPA）:「ソフトウェア開発データ白書2018−2019」、2019
3）マーク・パール:「モブプログラミング・ベストプラクティス ソフトウェアの品質と生産性をチームで高める」、日経BP、2019
4）エドワード・デ・ボーノ:「6つの帽子思考法―視点を変えると会議も変わる」、パンローリング、2015
5）市谷聡啓、新井剛:「カイゼン・ジャーニー たった1人からはじめて、「越境」するチームをつくるまで」、翔泳社、2018
6）西村直人、永瀬美穂、吉羽龍太郎:「SCRUM BOOT CAMP THE BOOK」、翔泳社、2013
7）Jonathan Rasmusson:「アジャイルサムライ−達人開発者への道」、オーム社、2011
8）平鍋健児，野中郁次郎:「アジャイル開発とスクラム〜顧客・技術・経営をつなぐ協調的ソフトウェア開発マネジメント」、翔泳社、2013
9）畑村洋太郎:「失敗学のすすめ」、講談社、2005