特集号
招待論文

グローバル分散開発におけるアジャイル適用

松下 望[†1]　若尾 正樹[†1]　岡本 修治[†1]

[†1] 日本アイ・ビー・エム（株）

ソフトウェア開発の中でアジャイル手法が広く取り入れられつつある．一方，大規模あるいは分散開発でのアジャイル適用には課題がある．本稿ではアジャイルの考えを発展させ，規模の大きい開発プロジェクトでも適用できるアジャイル・フレームワークである「Disciplined Agile Delivery」（以下 DAD）および「Scaled Agile Framework」（以下 SAFe）について紹介する．その適用例としてグローバルなソフトウェア製品開発の現場でのアジャイル手法とそれをサポートするツールの適用事例を取り上げ，大規模分散開発におけるアジャイル適用への課題と解決手法について説明する．

1. はじめに

近年，激化し続ける市場競争はソフトウェア開発においても例外でなく，刻々と変化する市場に即した開発が求められている．そのためには，技術的な実現可能性，スケジュールやリソース，対象となるユーザやマーケット，それらを取り巻く環境など，さまざまな不確定要素が存在する前提で開発を進めていく必要がある．

これらの不確実性を取り扱う開発手法として，近年「アジャイル開発」に注目が集まっている．たとえば，Dr. Dobb's Journalの調査[1]によれば，2012年時点で組織の86%がアジャイル開発に挑戦しており，その中の82%が何らかの形でアジャイル開発を成功裏に実践している．

その一方，同調査では，開発規模が大きくなればなるほど，成功裏に実践できなくなる傾向を示している．また，ドキュメントや設計は不要である，長期開発には向いていない[2]など，誤解が発生しているケースもある．

アジャイル開発手法を適用するには，まずアジャイルの本質を正しく理解することが必要である．

本稿では，第2章で，アジャイルの本質を説明するために，アジャイル開発の思想と代表的な手法を示す．第3章では第2章の内容を拡張したエンタープライズ・アジャイルの思想と代表的な手法を示す．第4章では第3章で示したエンタープライズ・アジャイル開発手法をIBMソフトウェア製品開発に適用した例を示し，第5章ではその適用時に把握した課題とその対応例について述べる．第6章ではエンタープライズ・アジャイルを広めていくための考察を記し，第7章でまとめとする．

2. アジャイル開発の思想と手法

2.1 アジャイル開発の思想

アジャイル（Agile）とは「機敏な」という意味の英語で，変化に対して機敏に対応し，顧客に価値あるソフトウェアを，迅速かつ継続的に提供することを目的としたソフトウェア開発方法論の総称である．

昨今，社会状況やマーケットの変動が激化し，業務が複雑化するのに伴って，ビジネス要件やシステム要件が刻々と変化している．しかし，ウォーターフォールに代表される重厚な開発プロセスでは，その変化に対して機敏に対応することが困難になってきている．

こうした状況を背景とし，変化への対応をうたう開発方法論としてアジャイルが提唱された．その中核を担うのが，2001年に発表されたアジャイル・マニフェスト[3]である．この中では，アジャイルの本質を語る上で基礎となる4つの価値宣言（図1）と12の原則が記載されている．

図2はウォーターフォールとアジャイルの固定要素と

図1　アジャイルの4つの価値宣言 [3]

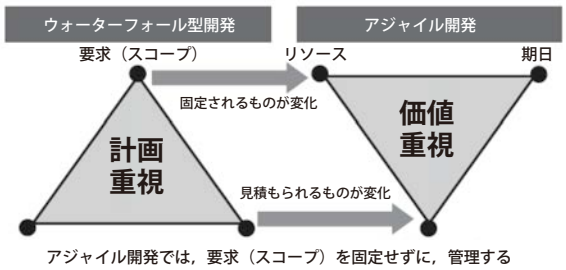

図 2　アジャイルの固定要素と変動要素 [4]

変動要素[4]を示したものである．ウォーターフォールではスコープを固定した上でリソースと期日を見積もるのに対し，アジャイルでは要件の変化に追随していくために最初からスコープを固定せずに進める．リソースと期日を固定させた上で，さまざまな前提条件や制約を鑑みながらスコープを都度調整する．優先度の高い要件から実装していき，顧客にとって価値あるソフトウェアを継続的に提供していく．

アジャイルでは，定期的に要件の優先度を見直すために反復型開発を取り入れているが，反復期間は2週間～4週間が望ましいとされている．

また，変化に機敏に対応していくためには，利害関係者と積極的なコミュニケーションをとることができる，協調性の高いチームを形成することが重要である．そのためには，メンバが自分自身をコントロールして自律的に行動し，目標に向かってチームの成長に貢献していく必要がある．

2.2　アジャイル開発手法

代表的なアジャイル開発手法には，「スクラム」[5]や，「エクストリーム・プログラミング（XP）」[6]，「リーンソフトウェア開発」[7]などがある．どの開発手法をプロジェクトで採用するかは，各手法の特徴を考慮して決めていく必要がある．

アジャイル・マニフェストでは，計画の遵守よりも変化への対応を重視している．アジャイルではこのような変化に機敏に対応するために，定期的に見直しを図ることができる反復型開発を取り入れている．アジャイルの代表的な開発手法であるスクラム（図3）を例にしてアジャイルの反復型開発サイクルを説明する．

スクラムでは，反復のことを「スプリント」と呼ぶ．まず，スプリントで開発すべき優先度の高い要件をスプリント・バックログに割り当ててスプリント計画を作成し，動作するソフトウェアを開発する．スプリントの終わりにはレビューにて利害関係者にデモを行い，要件とのギャップを洗い出した上でフィードバックをもらう．また，必ず振り返りを実施し，プロジェクト・チームの改善項目を洗い出し，次のスプリントに活かす．スプリントを繰り返すことにより，プロジェクト・チームの結束力を強め，持続可能なペースを維持しながら，継続的に顧客へ価値を届けることができる．

3.　エンタープライズ・アジャイルの思想と手法

3.1　スケーリング・ファクタ

前章で説明したアジャイル開発手法は，同一拠点で，かつ5～10名程度の小規模なチームで最も有効であるといわれている．しかし，規模の大きい開発でもアジャイル開発手法は適用できる．

ただし，その際には従来のアジャイル開発手法以外にも考慮すべき点がある．たとえば，大人数で地理的に分散した体制を組むような開発プロジェクトでは，国ごとの法的規定や文化的慣習を考慮した体制構築が必要になる．また，構築するシステムが，業界団体や監督官庁によって定められた規定・法令を遵守しなければならない場合もある．

IBMでは，このような規模に応じた変動要素をスケーリング・ファクタ（図4）としてまとめている．

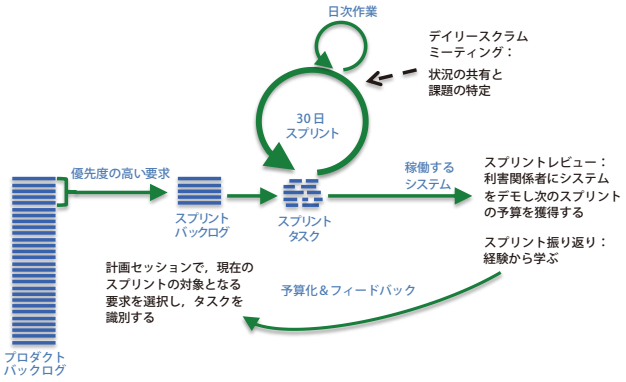

図 3　スクラムの開発サイクル [5]

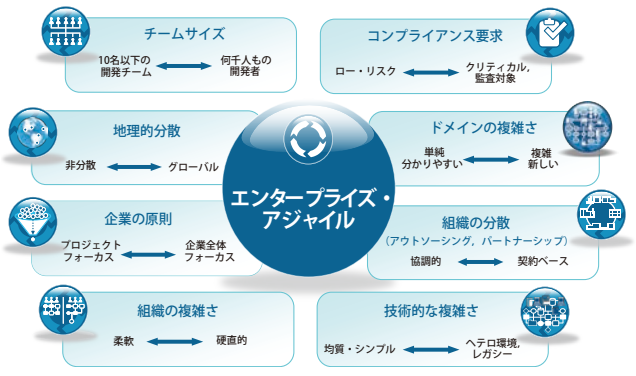

図 4　スケーリング・ファクタ

スケーリング・ファクタの考慮が必要なプロジェクトでは，意思決定に絡む利害関係者が多くなるため，スケジュール調整がしづらいなどプロジェクト運営のリスクが増大する．また，技術的な複雑さが増してくるため，品質が担保しづらいなど技術的なリスクも増大する．

これらに適切に対応していくためには，1回の反復期間中のチームの振舞いだけではなく，プロジェクトの開始から終了まで考慮した時間軸の中での意思決定ポイントやプロジェクトの運営を明確にした枠組みを用意し，利害関係者も含めた全体で合意しておく必要がある．

3.2　エンタープライズ・アジャイル開発手法

これらのスケーリング・ファクタを意識したエンタープライズ・アジャイル開発手法として，「Disciplined Agile Delivery」（以下，DAD）[8]，「Scaled Agile Framework」（以下，SAFe）[9]がある．

3.2.1　Disciplined Agile Delivery (DAD)

DADはアジャイル開発手法を中・大規模開発にも適用できるようにIBMが定義したアジャイル・フレームワークである．リスクと価値を天秤にかけながら，費用と期間の制約の中で高品質かつ有用なソリューションを定期的にデリバリーすることを目的としている．

DADの開発サイクル（図5）では，1回のリリースだけではなく，開発するシステムや製品のライフサイクルを意識した開発の流れを定義している．先述のスクラムは開発の進め方にフォーカスしているが，DADはこれをベースに，プロジェクト運営上の節目を明確にするためのフェーズとマイルストーンの概念を取り入れている．

中・大規模開発においては，適度なガバナンスを取り入れつつも，チーム全体の高い協調性と自律性，自分たちの手で改善を推進するアジャイル開発本来の良さを活かすことが重要となってくる．この二律背反する命題を両立させる上で「やるといいこと」をプラクティス集のかたちで豊富な実践経験をもとに体系化しているのがDADの最大の特長である．

3.2.2　Scaled Agile Framework（SAFe）

アジャイル開発の考え方を特定のソフトウェア開発プロジェクトだけでなく，企業活動全体にまで拡張したエンタープライズ・アジャイル手法として，SAFeがある．SAFeは企業活動そのものの俊敏さを向上させることを目的としており，その実現のために3階層のマネジメント・モデル（上位からポートフォリオ・レベル，プログラム・レベル，チーム・レベル）を提唱している（図6）．

ポートフォリオ・レベルは経営層もしくは事業部の責任者が担当する領域であり，ビジネス戦略に即して，それがビジネス上のニーズを満たすか，システム的な必要性が高いか，といった観点からどの投資テーマを重点的に実施するかの優先順位付けを行う．

次にプログラム・レベルではポートフォリオ・レベルで選定された投資テーマをどう実現するかを考える．通常，投資テーマは1つのスクラム・チーム（5～9人程度）が3カ月単位でリリースできる粒度よりも大きいため，その実現に向けた計画セッションを実施し，これから実施すべきことの優先順位を合意するとともに，複数のスクラム・チームを編成する．複数のスクラム・チームが協力しながら1つのシステムを作りあげていく上では，スプリントのサイクルを統一し，各チームが作成する成果物がいつ頃完成し統合できるのかといったチーム間の依存関係を計画・合意し，トラッキングすることが

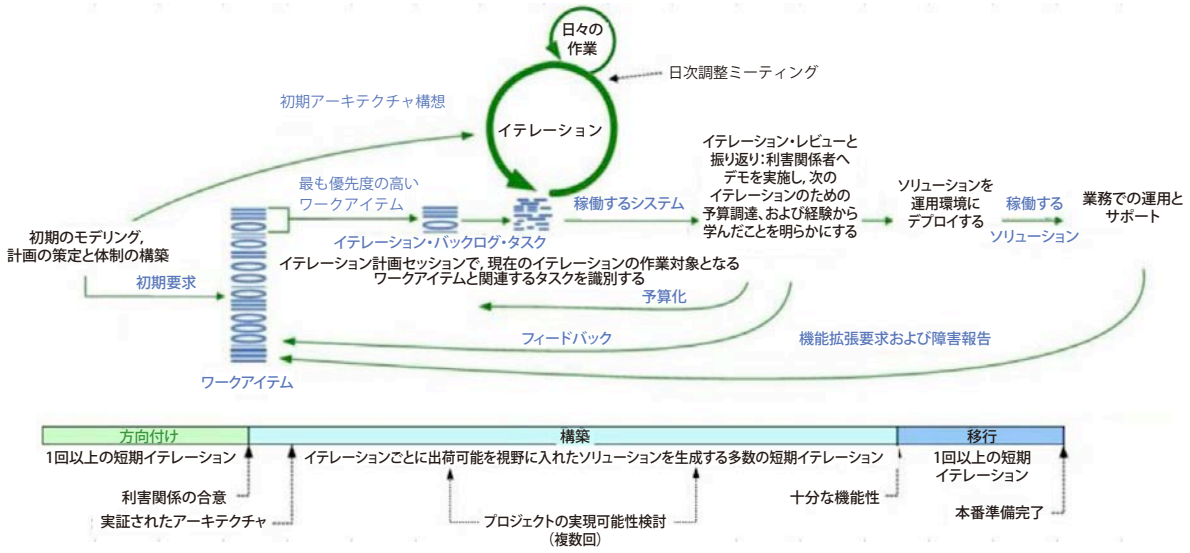

図5　DADの開発サイクル[8]

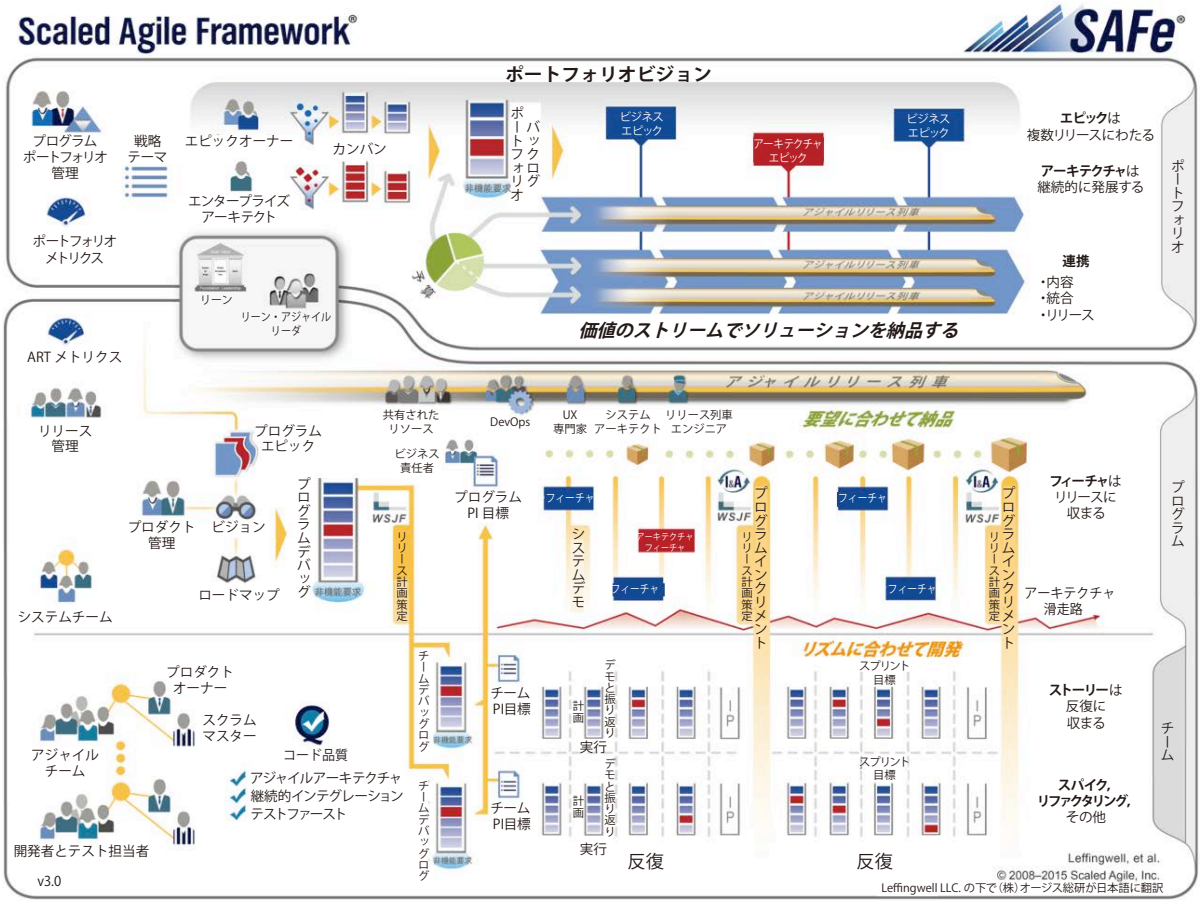

図 6　SAFe の全体像 [9]

重要となる. これはスクラム・オブ・スクラムと呼ばれる, スクラムマスター間のスクラムを通じて行う. また, システム全体のアーキテクチャを決めるアーキテクチャ・チーム, プログラム・レベルでのリリース時期の決定やリリースを行う上で必要な統合やテストの実施時期や方法を考えたり, チーム間のコンフリクトや課題をマネージするプログラム・オフィスと呼ばれる機能がこの階層に属する.

最下位のチーム・レベルはソフトウェア開発を行う各スクラム・チームであり, ここで規定されるプラクティスは通常のスクラムないしDADが提唱するそれとほぼ同等といえる.

3.2.3　DADとSAFeの位置づけ

DADとSAFeは, いずれもスクラム・ベースの構築作業にフォーカスしたアジャイル開発手法をエンタープライズに適用する上で必要な拡張を行っているが, そのベクトルは異なる.

DADは前述のスケーリング・ファクタや開発ライフサイクルを意識したフェーズの定義を通じ, アジャイル開発に適度な規律を持ち込むとともに, 開発作業の進め方を豊富な事例に基づくプラクティス集のかたちで具体的に説いている一方, 複数チームで開発を行う上でのプラクティスについてはあまり具体的には触れていない.

一方SAFeは, 特にプログラム・レベルにおけるプラクティスにおいて, 複数スクラム・チームを束ね, 1つのシステムを作っていく上での具体的な行動指針, 必要となるロール, チーム編成, 会議体などについて詳細に解説している.

まとめると, エンタープライズにおける複雑なプロジェクトの開発ライフサイクルの実践という観点ではDAD, 複数の開発チームを束ね1つの目標の実現方法の推進という観点ではSAFe, とそれぞれ得意領域がある. このように両者は相互補完関係にあり, エンタープライズ・アジャイルを実践する各主体が両者から自らに適したプラクティスを選択・適用することが重要になってくる.

IBMでは主にソフトウェア製品開発部隊においてその方法論を実践しており, またその実践を支援, 促進するために, DAD, SAFeの各種プラクティスに即したテンプレートを用意し, 開発作業を行う統合開発環境上でこれを活用している.

4. IBM ソフトウェア製品開発でのエンタープライズ・アジャイル適用

IBMでは，変化し続けるユーザの要求に機敏に対応するために，ここ10年の間にソフトウェア製品開発にアジャイルの考え方や手法を積極的に取り込んでいる．

アジャイルを推進するためにまず行われたことは数人からなるアジャイル推進チームが全世界の開発チームに対して，アジャイルの根本的な考え方の教育を行うことだった．アジャイル・マニフェストの理解から，リーンの考え方，XP，スクラムなどの手法，IBMにおけるアジャイルの適用方針などの教育を2日間かけて行い，さらに実際のプロジェクトへの適用の初期段階では，推進チームがアドバイザーとしてサポートを行うことによって，全社的に広めていった．その結果，現在では大多数のソフトウェア製品開発がアジャイルの考え方に基づいて行われている．

ここでは，1つのソフトウェア製品開発プロジェクトを題材に，スケーリング・ファクタに基づいてプロジェクトの特性を示し，その特性に基づいてどのようなプラクティスを適用しているかを説明する．

4.1 スケーリング・ファクタに基づいた開発プロジェクトの特性

スケーリング・ファクタから，本プロジェクトのプロジェクト特性をいくつか示す．

4.1.1 チームサイズ

IBMのソフトウェア製品開発では，数十人から100人程度の開発者が参加するケースが多い．これは一般的なアジャイルで適当とされる10人程度と比較すると大規模だといえる．本プロジェクトのサイズも100人程度である．

4.1.2 地理的分散

参加メンバが地理的に分散しているケースが多く，極端な例では，アメリカ，ヨーロッパ，アジアから開発者が参加して1つの製品を開発することも珍しくない．本プロジェクトでも，アメリカの複数の場所，ヨーロッパの複数の場所，日本，中国からのメンバが主に参加している．

4.1.3 技術的な複雑さ

本プロジェクトでは，顧客に提供するソリューションを実現するために，複数の製品を連携する必要があるものになっている．規模が大きくなり，複数製品間での連携が必須であることから，そのアーキテクチャも複雑化している．

4.2 プロジェクト特性に基づいたプラクティスの適用

4.1節で述べたプロジェクト特性に対して，本プロジェクトではエンタープライズ・アジャイル開発手法を参考にしながら以下のプラクティスを導き出し，適用している．

チームサイズ

- チーム構成の階層化
- チーム間で同期した反復
- ステークホルダ要求の共有
- ツールによるルールの適用

地理的分散

- コラボレーション
- ソースコード管理とビルド

技術的な複雑さ

- 方向付けと移行フェーズ

4.2.1 チーム構成の階層化

100人規模のプロジェクトにアジャイルを適用する必要があるため，プロジェクトは初期段階でコンポーネント化を行い，各コンポーネントに責任を持つ複数のチームで構成し，各チームがアジャイルを運営する形態をとる．チーム間での整合性をとるため，スクラム・オブ・スクラムを適用している．また開発チームとは別に，製品レベルでの統合テストを行うチームや，製品レベルでユーザインタフェースの統一性に責任を持つチームが存在し，製品レベルでの品質を確保している（図7）．

4.2.2 チーム間で同期した反復

各チームは2週間から4週間の反復を行うが，反復の完了時には製品レベルで統合した動くソフトウェアを提供することが求められるため，プロジェクトを構成するすべてのチームが同じ反復スケジュールに従う．反復の中で統合テストチームによる製品レベルでの統合テストも実行され，反復の完了時には統合された動くソフトウェアが生成される．結果，ステークホルダが実際に試した結果から得られるフィードバックを真の要求として取得することができる．

4.2.3 ステークホルダ要求の共有

チームサイズが大きくなり，地理的に分散すれば，ス

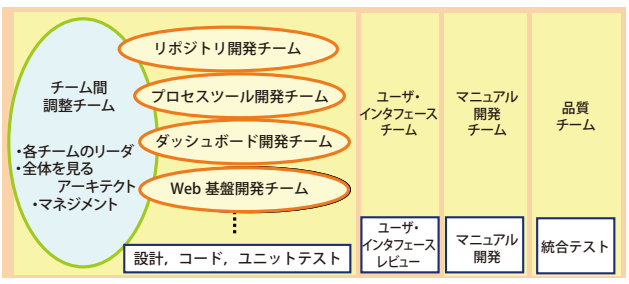

図7　チーム構成

テークホルダからの要求を全メンバで共有することは容易ではなくなる．

本プロジェクトでは，社内外のステークホルダからの方針や要求，フィードバックなどは，その粒度に応じて，SAFeでいうポートフォリオ，プログラム，チームレベルのバックログに追加され，プライオリティ付けされた上で，重要度の高いものから順に次のレベルへと詳細化が進められ，最終的にチームレベルでのストーリーからタスクまでが作成される．

各レベルのバックログからタスクは，その間の関係も含めて開発ツールで管理されており，いつでもトレースすることが可能になっている．開発者は日々タスクを実施していくが，タスクの目的とするストーリーや，そのもとのステークホルダからの要求を確認することが可能になっていることは，プロジェクトの方向性をチーム全体で共有するという観点で有効に機能している．

4.2.4　ツールによるルールの適用

規模が大きくなるほど，各メンバにプロジェクトのルールを適用の徹底が困難になり，品質の低下につながる可能性がある．

本プロジェクトではツールによりルールを自動的に適用することでそれを回避している．たとえば，後半の反復では，チームリーダの承認なしではソースコードのチェックインがシステム的にできないようにしている（図8）．

4.2.5　コラボレーション

アジャイルにおいてチーム内での情報や状況の共有を行うことは重要である．

本プロジェクトでは，多くのメンバが参加し，地理的に離れたメンバ間でも密な共有を行うため，電話会議によるスクラム・ミーティング，チャットによるリアルタイム・コラボレーションに加え，ツールでバックログやストーリー，タスクを管理し，それぞれにコメントをメンバが付加することによって，同じコンテキストでの状況の共有を行っている．さらに，入力されたタスクの状況を要約してチームのダッシュボード上で可視化することによって，チームの誰もがチームあるいはプロジェクト全体の状況を共有できる（図9）．

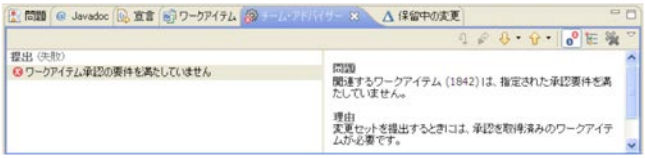

図8　ツールによるルール適用

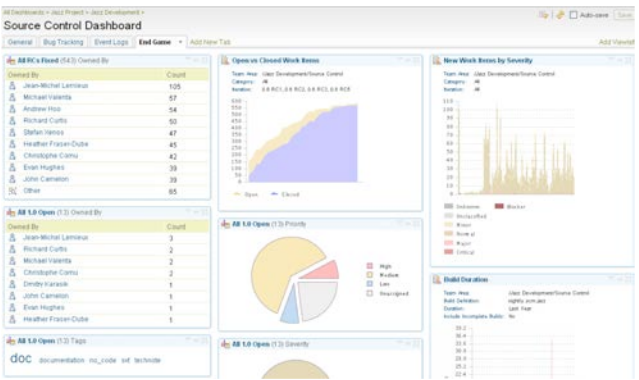

図9　チーム・ダッシュボード [10]

4.2.6　ソースコード管理とビルド

グローバルに分散した複数のチームでソフトウェアコードを共有，継続的にビルドを行い，常に動くソフトウェアを作成するため，ソースコードは，グローバルで共通にアクセスする構成管理ツール上で管理している．ビルドはまずチームごとに行い，そこで成功した安定したソースコードを統合ビルドに移行し，最終的な製品レベルのビルドが常に成功するような工夫を行っている（図10）．

4.2.7　方向付けと移行フェーズ

製品横断での連携や製品間でベースとなるフレームワークが必要となるケースも多いため，リリースの最初の数日から数週間にはベースとなるアーキテクチャ構築を行い，プログラムレベルでのバックログを決定する（方向付けフェーズ）．最終的なリリースの前には，テストと見つかった問題の修正，ドキュメント作成だけの反復を数回行い，品質を確保している（移行フェーズ）（図11）．

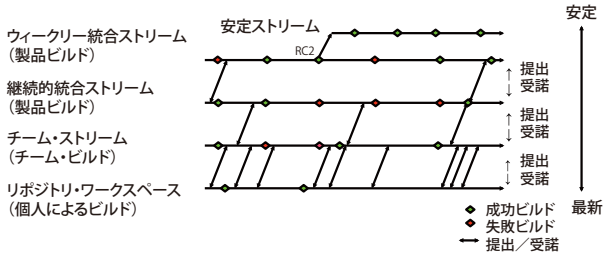

図10　階層化されたビルド

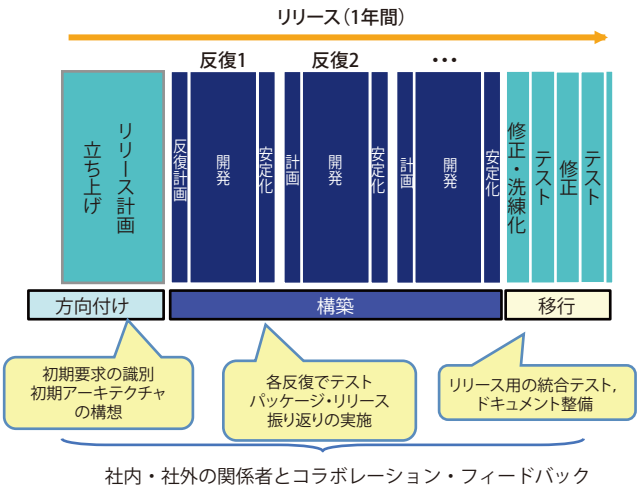

図11　方向付けと移行フェーズ

5. エンタープライズ・アジャイルの課題と対応例

第4章でIBMのソフトウェア製品開発におけるエンタープライズ・アジャイルの適用事例を紹介した．ここでは，エンタープライズ・アジャイルを適用した際に気付いた課題とその対応例を記述する．

5.1 プログラムからのチーム管理

5.1.1 課題

エンタープライズ・アジャイルにおいては，チームを複数に分け階層化して運営していくことになる．各アジャイルチームの運営はチームに委譲し，チームの主体性を重んじることが望ましい．一方SAFeにおけるプログラムレベルのチームは全体の進捗やチーム間の依存関係を明確にするため，常に各チームの状況を把握する必要がある．しかし，自主的に運営されるチームは，そのスキルレベルや運営の違いなどにより，全体を管理していくことは容易ではない．チーム間でスケジュールに差が出たりすることは，プログラム全体のスケジュールにも影響を及ぼすため，全体をまとめて運営していくことは重要な課題となる．

5.1.2 対応例

第4章で述べたIBMのケースのように，ツールを活用し，各ストーリーやタスクをプログラムレベル全体でひも付けて管理し，進捗や依存関係をどのレベルでも可視化するようにすることは1つの解決方法として有効と考えられる．

特定チームの遅れの原因は，見積の精度の問題，技術的リスクの顕在化の2つに大別される．前者については不確定要素がつきものの開発作業において完全に回避することは不可能だが，チームにより楽観的／悲観的な傾向やバラつきの大きさは表れるので，時系列的にその傾向を上位レベルで測定し織り込んでおき，プロジェクト全体で軌道修正を試みる．後者については次節で述べる通り，チーム内もしくはチーム間の横の協業で対処を試みる．

5.2 プログラムでの反復スケジュールの維持

5.2.1 課題

プログラムすなわち複数チームで開発を行う場合，各チームが同じ反復スケジュールを守り，コミットした反復スケジュールで動くソフトウェアをつくることが条件となる．アジャイル開発ではリソースとスケジュールを固定しスコープを可変とするのが原則だが，チーム間の依存関係がある場合には1つのチームのスコープの変更は全体のスケジュール影響を及ぼすため，前述の制約が生じる．

この制約のもと円滑に開発を進める上では，お互いのチーム状況を偵察し調整する能力や，見積りの確度が重要になってくるが，総じてこれらは個々人の能力や性格に大きく依存する性質がある．

5.2.2 対応例

チーム間での人的リソースの柔軟な組み替えを行うことによってチーム間の差を吸収し，全体に影響を及ぼさないようにし，プログラム全体でのスケジュールおよびリソースは固定する．

依存関係の識別・調整は一般にはスクラムマスターの役割だが，単独チームによるアジャイル開発と比べてこの作業の負荷は大きく，スクラムマスターの疲弊につながりやすい．そのような場合は，スプリントごとにチームメンバ間でリーダ役を持ち回りするなどの方法で負荷を分散させる．たとえば序盤のスプリントでは上流の経験が豊富なメンバ，終盤の安定化スプリントではテストや品質に強いメンバがスクラムマスターを担当するなど，得意分野を考慮した持ち回りが有効である．

5.3 進化型アーキテクチャ

5.3.1 課題

複数チームで整合性をとってプロジェクトを運営していくためには，アーキテクチャは方向付けフェーズで初期アーキテクチャとして固める必要がある．初期にアーキテクチャが固まることによって，検証に必要な最低限の機能を持った製品をいち早く実現でき，リスク軽減にも繋がる．

ただし，アジャイルにおいては，要求の変化に応じてアーキテクチャも進化させていく必要がある．アーキテクチャの変更は各チームへ影響を与えるため，全体としての整合性をとることが課題となる．

5.3.2 対応例

アーキテクチャが変更となる場合，プログラムレベルのアーキテクチャ・チームがアーキテクチャの変更に対応するための特殊なタスクを作成し，その階層化したサブタスクを各チームに伝搬し，各チームが対応したかどうかを確認するような仕組みを構築することにより，全体の整合性の維持を確保する．

5.4 分散開発におけるコミュニケーション

5.4.1 課題

アジャイルでは，チーム一人ひとりがチームの状況に応じて柔軟に対応し，チーム全体でのパフォーマンスを最大限にすることを目指しているため，チーム内でのコミュニケーションはとても重要になる．しかし，エンタープライズ・アジャイルにおいてはチームメンバが地理的に離れ，場合によっては時差も違う地域からメンバが参加する場合もあり，コミュニケーションを円滑に行うことが課題となる．

5.4.2 対応例

地理的に離れていても，重要なイベントとしての一日一度のスクラムミーティングや，反復完了時の振り返りのような全員が参加する会議は開催する．そのために第4章のコラボレーションの部分で記述したようにツールの活用は非常に有効になる．常に全員がチャットツールを起動し，すぐにコミュニケーションをとれるようにし，必要に応じて電話会議を即始められるようにする．できればお互いの顔を見ながら話ができるビデオ会議のような仕組みがあるとなお良い．

時差がある場合には，リアルタイム・コミュニケーションを行う時間帯が限定されてしまう．それを補完する上でツールで管理された項目（要求項目やタスク）にコメントを書き込むタイプのコミュニケーションは有効である．たとえば日本時間の夕方コメントを書き込んでおき，アメリカ時間の昼，アメリカの開発者がコメントに対して返信をすれば，日本には次の日の朝に回答が来ていることになる．メールでのやりとりと違い，項目そのものに対して履歴が残るため，あとでほかの人間がその項目を理解することにも役に立つ．

しかし，やはりお互いの顔を見て話すこと以上のコミュニケーションはない．現実的にはメンバで一度も顔を合わすことなく仕事を行なわざるを得ないこともあるが，プロジェクト開始時に一度でも顔を合わせる機会を持つことは，確実にその後の作業効率を上げるものと考える．

5.5 チーム・メンバのスキル平準化

5.5.1 課題

エンタープライズ・アジャイルを適用するような規模の開発では，多様なチーム・メンバがいるためスキルのばらつきが発生しやすい．特に分散型開発の場合，ベースとなるエンタープライズ・アジャイルの考え方やプラクティスをチーム間で共有しておかないと，コミュニケーションが円滑に行われず，開発がうまくいかないリスクが高くなる．

5.5.2 対応例

メンバがプロジェクトに参加する前に，アジャイルに関する最低限の知識やスキルを習得する必要がある．エンタープライズ・アジャイルを適用する場合，チームごとに標準教育を用意するのではなく，プログラムレベルで用意するのが望ましい．

ただし，標準教育を受講しただけでは不十分なケースが多く，実体験を元に自分たちのやり方にあわせてプラクティスを改善していくのが大切である．

また，アジャイルの経験が乏しいチームが存在する場合，アジャイル経験者がチームをコーチングしながら進めるのが望ましい．

6. エンタープライズ・アジャイルを広めていくためには

以上のように，DAD，SAFeをベースとしたエンタープライズ・アジャイルの概念と適用事例，および課題と対応例を紹介した．

ただし，まだ世の中にエンタープライズ・アジャイルが浸透しているとは言いがたい．エンタープライズ・アジャイルを広めていくためには，以下のようなことを考慮していく必要があると考える．

6.1 成果物中心から価値中心へ

不確定要素が多い中で開発を進める場合，最初からすべての成果物を規定するのが困難であるのにもかかわらず，成果物の作成に注力してしまうケースがある．特に，大規模開発ではコミュニケーション・パスが多岐にわたるため，決定事項を伝えるための中間成果物が数多く必要となるケースが多い．

しかし，アジャイルソフトウェア開発宣言に記述されているとおり，包括的なドキュメントよりも動くソフトウェアに価値がある．これは，開発規模にかかわらず変わらない，アジャイルの価値の1つである．

コミュニケーションを主体としながら常に顧客の価値を注視し，中間成果物の作成に工数を取られないようにするのが大切である．また，顧客の価値を見極めるための要求開発を全ステークホルダで行うのが望ましい．

6.2 プログラムレベルからのアプローチ

エンタープライズ・アジャイルは適用するプロジェクト規模が大きくなるため，全社的なアプローチが必須で

ある．まず，会社の経営に直結するポートフォリオレベルを整備し，その戦略をプログラムレベルやチームレベルに落としていく，トップダウンアプローチが望ましい．

ただし，すべての会社の経営者がトップダウンアプローチでエンタープライズ・アジャイルを浸透できるとは限らない．このような状況では，スケーリング・ファクタをできるだけ最小限にしながらプログラムレベルとチームレベルを整備し，その結果をポートフォリオレベルに反映させていく，ボトムアップアプローチをとるのも手である．SAFeにおいても，その効果を享受するための最小セットとしてプログラムレベルとチームレベルの活動に焦点を当てている[11]．

6.3　エンタープライズ・アジャイルの理解浸透

第3章で述べたように，エンタープライズ・アジャイルを実践するためには，プロジェクト特性に応じたスケーリング・ファクタを正しく認識し，その結果に応じて適用するプラクティスを決定し，調整するのが望ましい．

そのため，ベースとなるアジャイル開発手法のプラクティスだけではなく，エンタープライズ・アジャイル固有の複雑さや，それに対応するためのプラクティスも合わせて理解する必要がある．

また，開発チーム内だけではなく，多岐にわたるステークホルダに対しても，経験者がエンタープライズ・アジャイルの価値や原則を啓蒙していき，共通理解を持ちながら進めていくことも大切である．

7. おわりに

本稿では，IBMのソフトウェア製品開発における固有のスケーリング・ファクタとそれらを解決するためのプラクティス例を紹介した．また，これらのプラクティスを本番プロジェクトに適用した結果，真の要求の獲得と実装，製品の品質向上，および製品開発メンバのモチベーション向上が見られた．

これらのプラクティスは反復ごとに行う「振り返り」でのメンバによる自発的なアイディア出し，実践，改善に基づくものであり，開発活動のコンテキストやチームの特性に応じ有効なプラクティスは変わることに留意されたい．つまり「どうやるのか」ではなく，「なぜそうする（と良い）のか」をメンバ全員で継続的に考え，実行に移すことが重要である．

上記を踏まえ，本稿第4章および第5章で述べたプラクティス例がエンタープライズ・アジャイル実践の一助となれば幸いである．なお，その普及を図っていく上では，第6章で述べた観点からの理解浸透やアプローチの工夫が不可欠となろう．

参考文献

1) Dr. Dobb's Journal (DDJ)：Surveys Exploring The Current State of Information Technology Practices, http://www.ambysoft.com/surveys/（2016 年 4 月 13 日現在）
2) JaSST：アジャイル開発の勘違い, http://jasst.jp/archives/jasst06w/pdf/C3-2.pdf（2016 年 4 月 13 日現在）
3) Manifesto for Agile Software Development : Manifesto for Agile Software Development, http://agilemanifesto.org/（2016 年 4 月 13 日現在）
4) Leffingwell, D.：アジャイル開発の本質とスケールアップ，（株）翔泳社（2010）．
5) 西村直人，永瀬美穂，吉羽龍太郎：SCRUM BOOT CAMP THE BOOK，（株）翔泳社（2013）．
6) Beck, K.：XP エクストリーム・プログラミング入門：ソフトウェア開発の究極の手法，（株）ピアソン・エデュケーション（2000）．
7) Poppendieck, M. and Poppendieck, T.：リーンソフトウェア開発――アジャイル開発を実践する 22 の方法，（株）日経 BP 社（2004）．
8) Ambler, S. W.：Disciplined Agile Delivery エンタープライズ・アジャイル実践ガイド，（株）翔泳社（2013）．
9) Leffingwell. D.：アジャイルソフトウェア要求，（株）翔泳社（2014）．
10) jazz.net: Jazz Community Site, https://jazz.net/（2016 年 4 月 13 日現在）
11) First Thing's First: Essential SAFe? http://www.scaledagileframework.com/first-things-first-essential-safe/（2016 年 4 月 13 日現在）

松下 望（非会員）matusita@jp.ibm.com
入社以降，ソフトウェア開発エンジニアとして製品開発に従事．現在は開発支援ツールである Rational 製品の導入支援やコンサルティングに従事．PMI 日本支部アジャイル・プロジェクトマネジメント研究会でも活動し，PMI のアジャイル資格である PMI-ACP®を保有．

若尾 正樹（非会員）wakao@jp.ibm.com
1990 年に入社以来現在に至るまでソフトウェア製品開発に従事．OS 開発，コンシューマ向けソフトウェア開発，Web アプリケーション開発を経て，現在はアジャイルも含めたソフトウェア開発支援環境の開発に携わる．

岡本 修治（非会員）osyuhji@jp.ibm.com
大規模 SI プロジェクトにおける開発，開発手法やプロジェクトマネジメント手法とツールの社内展開に従事．現在は DevOps 関連の製品技術に携わる一方その根底を成すアジャイルやリーン手法のコンサルティングも行う．認定 SAFe プログラム・コンサルタント（SPC）．

採録決定：2016 年 4 月 13 日
編集担当：上條浩一（(株）日本アイ・ビー・エム）