アジャイルの導入と本質
―開発プロセスから学習する組織活動へ―

平 井 直 樹

The introduction and essence of agile:
From development process to learning organizational activities

HIRAI, Naoki

ソフトウェア開発では，これまでの計画とその遂行，管理を中心としたウォーターフォールからアジャイルへとその開発手法が移りつつある。このアジャイルは，ソフトウェアを開発するうえで，より良いものを作るため，変化に対して迅速に継続的に適応できるようにするというものである。

アジャイルの本質は，顧客にとってより良いものをつくるための顧客との共創を前提とした「反復」活動を通じた組織的な学習であり，成功や失敗から学び，カイゼンしていく活動である。そのためには，能動的に動ける自己組織化，そして機能横断的なチームに成長していくことが重要であり，こうした取り組みはソフトウェア分野に限定されず，自動車などの他のプロダクトやマーケティングや人事などにも適用されはじめている。アジャイルはもはや単なるソフトウェアの開発手法や開発プロセスではなく，共創の場の中でイノベーションを起こしていく学習する組織活動として捉えることが可能であり，ビジネスを作る人々の活動の根底となり得る。

キーワード：アジャイル（agile），スクラム（scrum），ウォーターフォール・モデル（waterfall model），共創（co-creation），反復（iteration），ふりかえり（retrospective）

1．はじめに

ソフトウェア開発の世界では，これまでの計画とその遂行，管理を中心としたウォーターフォール・モデルと呼ばれる開発手法に代わるものとして，アジャイルと呼ばれる開発手法が注目を集めてきた。

アジャイル（agile）という単語は，「すばやい」や「俊敏な」を意味する。これまでソフトウェア開発では，1970 ～ 1980 年代の市場の変化や成長が比較的安定していた時代のもと，定型化，標準化したプロセスが求められてきた（Cusumano, 1991, 2004）。

このアジャイルは，日々新しいソフトウェアをリリースし，他社が追随できないサービスを提供し続けることにより，激しい変化を伴う競争環境に対応しようとするものである。特に，解決すべき問題が複雑であったり，解決方法が不明であったりするなか，イノベーションを起こすために，もっと良いプロダクトを作るにはどうすればよいのかということを追求するものでもある。

アジャイルは，硬直化した日本のソフトウェア開発を根本から変えるものとして期待されてきたが，これまでの方法と大きく異なることによる導入の難しさや，アジャイル自体の実施経験の少なさから失敗することも多く，なかなか浸透していない状況にある。一方，アジャイルが既にソフトウェア開発の基本となっている欧米では，このアジャイルを中心としたプロセスをソフトウェア開発の分野に限らず，製品開発のほか，組織運営などの分野へも活用され始め

ている。

本研究では，このアジャイルについて，ソフトウェア開発における特徴を整理するとともに，他の分野に適用するための要因を検討する。

2. 開発プロセスの比較

2.1 アジャイルのプロセス

アジャイルが，その発祥元であるソフトウェア開発でどのように行われているのかを確認する。

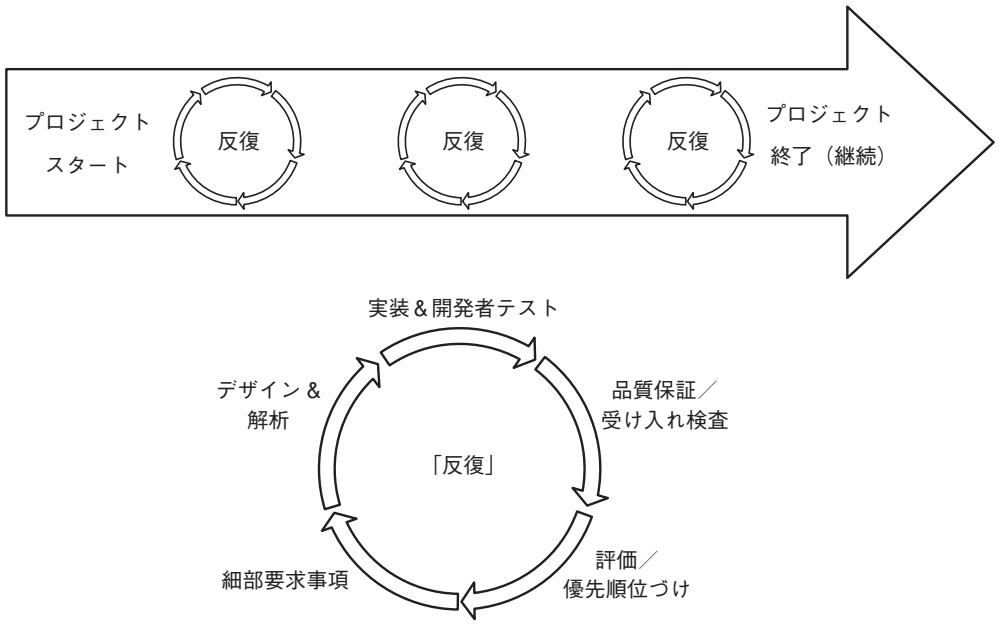
アジャイルは，正確には特定の手法を指しているのではなく，スクラム（Scrum）[1] や XP（extreme programming）と呼ばれる，幾つかの軽量なソフトウェア開発プロセスの総称であり，後述するウォーターフォール・モデルと呼ばれる多くの手順に従って進んでいく重量級な開発手法と比較される（妹尾，2001）。

アジャイルのうち，代表的なスクラムと呼ばれる手法のプロセスを図 1 に示す。

図 1 の上部の左から右へ流れる矢印は，アジャイルの全体の作業フローを表している。アジャイルは，この作業フローの中で，何度も「反復」（イテレーション）を繰り返す。下部の円を描く矢印が，この「反復」の内容を表している。「反復」の中では，顧客からの要求事項に基づくソフトウェアの作成，作成したソフトウェアの検査や評価，評価に基づいた新たな要求事項の提示，要求事項のデザインや解析といった作業を PDCA サイクルのように回している。

アジャイルの開発方法は，ソフトウェアの仕様について厳密な決定をせず，開発プロセスを進める中で仕様を擦り合わせ，ある程度決まった部分だけを先に開発していくスタイルである。このため，アジャイルは厳密な仕様を必要とする銀行などの金融機関の大規模な基幹システムといったソフトウェア開発には適していないが，スマートフォンのアプリケーションや EC サイトの構築といった中小規模のソフトウェアなどに適しているとされる。

ウォーターフォール・モデルに対し，アジャ



出所：“Scrum Reference Card”をもとに筆者作成。

図 1　アジャイル開発手法（スクラム）

イルではすべての機能を一度に取り込むのではなく，幾つかの機能を選択して開発を行う。さらに，その開発作業も１週間から１か月程度の非常に短い期間で反復して行い，それぞれの反復期間の終了ごとにソフトウェアを本番稼働させることを目指している。

さらにこの「反復」では，要求事項の中で優先度の高いものからできるだけ早く作成する。少しでもできあがったソフトウェアをユーザーに確認してもらうことで，早期にフィードバックを得て，再び「反復」に戻るのである。

アジャイルは，フィードバックといった顧客の参画の度合いが強く，そのため，人と人とのコミュニケーションやコラボレーション，共創を重視している。さらに，変化を前提とし，開発前の仕様の固定を前提としておらず（独立行政法人情報処理推進機構，2011），完成してからソフトウェアを動かすのではなく，ある程度動くソフトウェアを成長させながら作成する，反復・漸進型な部分が大きな特徴であり，複雑な問題を解決する際の創造性と順応性を持ったチームワークを重視している。

こうしたアジャイルのアプローチに共通するのは，ソフトウェアに対する顧客のニーズを理解することは難しく，その技術の移り変わりも早いため，そのような複雑なソフトウェアを前もって完全に設計することは無理があるという点である。ユーザーからのフィードバックや市場の変化に応えるような開発中の設計の変更は，成功に結び付くようなものであれば，必ずしも悪いものではない（Cusumano，2004）。

ただし，反復・漸進型な性格上，ユーザーからの要望一覧は，最後まで追加，更新されていくため，最終的にすべて解消できるわけではない。しかし，最後の最後まで少しでも良くしていくことがアジャイルの目的なのである（西村・永瀬・吉羽，2013）。

2.2　ウォーターフォール・モデルのプロセス

次に，アジャイルと対比されるウォーターフォール・モデルについて，確認する。

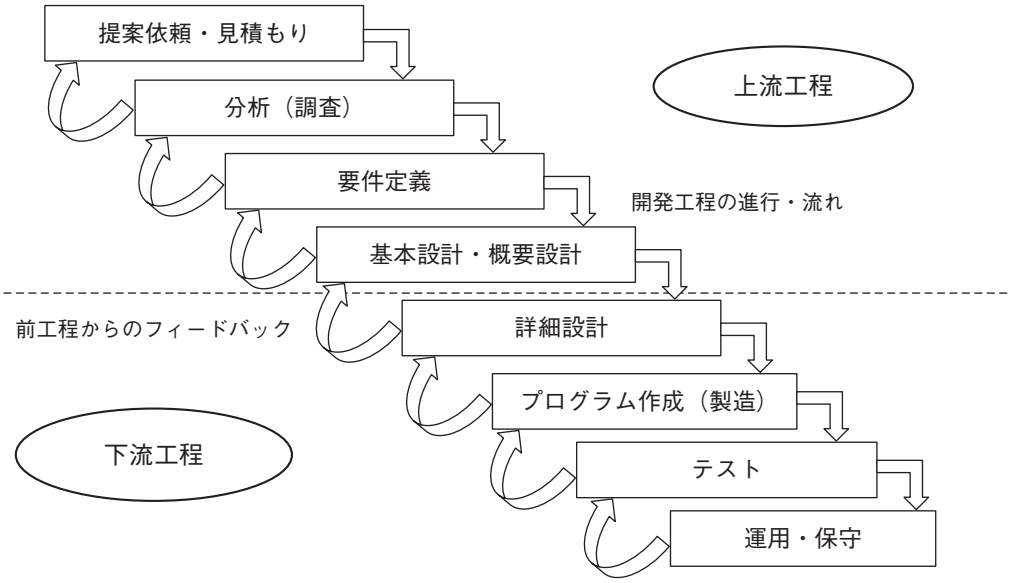
日本のソフトウェア開発では，アジャイルはまだあまり採用されていない。どちらかといえば，ウォーターフォール・モデルを採用することが多い。ウォーターフォール・モデルは厳格な手順を守ることを重視しており，正しいプロセスさえ踏めば正しい結果が得られることを想定している。つまり，顧客からの要求とそれに伴う要件や仕様が，プロジェクトが始まった時点で既に決まっており，そのプロジェクトが完了するまでそれらが変わらないことを前提としているのである。

この開発手法の大きな特徴は，原則として作業の順番が決まっており，見積もりや分析といった作業から順番に工程が進んでいき，必ず前の工程が完了してから次の工程へ進まなければならない。この工程の遡りが発生しないように，水が滝を流れ落ちるような手順となっているため，ウォーターフォール・モデルと呼ばれている。

このソフトウェア開発におけるウォーターフォール・モデルを図２に示す。

上流工程を重視し仕様変更を防ぐウォーターフォール・モデルは，製造業をモデルとした日本のソフトウェア開発に広く浸透してきた。モデルとなった製造業や建築業では，仕様変更を極力防ぐ方法が取られてきた。製品の仕様の決定や概念設計などの上流工程が完成してから，実際の製造などの下流工程へと入る方法である。その理由として，製造業や建築業では，一度製品の製造や建設作業に着手すると，工程を後戻りした場合，設計図の変更，金型の再作成，原材料・資材の再調達などが発生し，莫大なコストが発生してしまうためである。例えば，家屋やビルを建てる場合，一度工事が始まってからの大きな変更は難しい。このため，下流工程から上流工程への後戻りを極力行わないような手順になっている。

日本のソフトウェアは，元々コンピューターなどのハードウェアメーカーによって作成され



出所：Royce（1970），小椋（2013）をもとに筆者作成。

図２　ウォーターフォール・モデル

てきた経緯があり，1990 年代まで，そうした高価なハードウェアに付随するおまけとしてソフトウェアも一緒に作成されてきた。そのため，ソフトウェアの開発は，ハードウェア，つまり製造業の品質管理や標準化などの開発手法がそのまま適用されてきた影響が大きいのである（高橋，2010）。

このウォーターフォール・モデルは，前の工程に戻らないことを前提としているため，ソフトウェア開発プロジェクトの全体を把握することができ，スケジュールの立案や資源配分，進捗状況の理解が容易となる。

ウォーターフォール・モデルとアジャイルの開発スタイルの比較を表１に示す。

ウォーターフォール・モデルは，開発工程を複数に分割し，作業を定型化，開発プロセスを標準化した単純労働とすることで高い品質を保持してきた。さらに定型化，標準化されていることで，その作業を外部企業等にアウトソーシングするような分業も可能となる。それゆえ高い信頼性が要求されるようなシステム開発に有効とされてきたのである。

アジャイルは，開発のスピードが上がるだけでなく，ある程度完成したものが見えてくるためユーザー企業の確認も行いやすく，仕様変更にも対応しやすいなどの利点がある。一方で，ウォーターフォール・モデルは，工程の遡りが発生しないように必ず前工程が完了することが前提となっているため，それぞれの作業工程が長期化してしまうなどの問題がある。特に 2000 年以降，インターネット環境が整備されビジネスのスピードが上がるとともに，ソフトウェア開発にもスピードが求められており，開発するソフトウェアによってはアジャイルが優先的に採用される場合もある。

2.3　試行錯誤的プロセス

ソフトウェアは，ユーザー企業が求めるものとともに変化し，より高度に複雑化してきており，処理の自動化や制御といった画一的で機械的なものから，意思決定や戦略策定の支援といった個々の企業に合わせたものへと成長してきた。ビジネスを取り巻く環境の変化により，ソフトウェア開発も開発プロセスを順に追って

表１　ウォーターフォール・モデルとアジャイルの開発スタイルの比較

	ウォーターフォール・モデル	アジャイル
基本スタイル	予測型	適応型
計画	100％正確な計画を最初に立て，それに従う	正確な予測は不可能
変更	変更は（基本的に）しない	ユーザーの要望を満たし， 良いものを作るために必要なもの
要求	最初に確定させて吸い上げる	変化するものとして，随時対応する
開発目的	最初の要求通りソフトウェアを作り上げること	変化するユーザーの要望を満たすこと （より良いものをつくる）
人的要因	人為的な部分を排除，管理する	人の要素を活かす
環境	安定した環境が前提	変化の多い環境
開発人数	数人から数百人，数千人 （自社で足りない分は下請け企業等を導入）	10 人未満 7 人前後 （コミュニケーションが取れる範囲内）
開発体制	階層型	チーム型（少人数）
マネジャー （指揮者）	必要 （マネジャーやリーダー）	不要 （自分たちで能動的に動く）
開発期間	数か月から数年間	1 ～ 4 週間の繰り返し
製品の リリース	1 回のみ	最短最小のリリースを繰り返していく

出所：Glass（2006），平鍋・野中（2013）をもとに筆者作成。

いくような手法では対応できなくなってきた。

ソフトウェアの開発にとって，今までにない問題や新しい領域に取り組んでいくような革新的なソフトウェアを作る部分は，他の企業に対して差別化できる部分でもあり，競争優位を生み出す価値のある部分となりうる。

顧客ニーズの変化に柔軟にかつ迅速に対応し，イノベーティブなソフトウェアを作成するという行為は，組み立てラインのように作業分割してできるような簡単なことではない（Bean, 2005）。設計やデザインは初めから完成するものでなく，作業工程を進めていく中で何度も練り直すことが重要となる。

ウォーターフォール・モデルは，試行錯誤を経て魅力的な機能を持つようなイノベーティブなソフトウェアを開発する方法としては最良ではない（Cusumano, 2004）。現実には，あらゆる状況を想定したプログラムを作成することは難しく（Ferguson, 1992），時間をかけても完璧な計画を立てることは不可能であり，そのような方法ではユーザーが満足するシステムを作成することはできない。

特にソフトウェア開発作業では，作業者の専門知識やノウハウ，経験，創造的な問題解決といった知識労働としての側面が重要な役割を果たすことになる（平井，2018）。質の高い，革新的なソフトウェア開発には，開発に従事するエンジニアの知的あるいは創造的な能力の活用を阻害しないような反復，漸進的な開発プロセスの編成を実現する必要がある。

ソフトウェアの開発は，ユーザー企業の要望を製品化するために，アジャイルのように試行錯誤を繰り返しながらその要望を取捨選択して具体化する必要があり（大日方，1971），デザイン思考に基づく製造業の製品設計に近いものといえ（妹尾，2001；Cusumano, 2004），そこに現れてくる不確実で複雑な問題に対して，発見と解決のサイクルを繰り返していく高度な知識創造活動としての側面を有するのである。

3. アジャイル（スクラム）の導入

3.1 自己組織化と機能横断的

ここまで，ウォーターフォール・モデルとアジャイルの開発プロセスの特徴を比較，検討してきた。次にアジャイルが有する組織的特徴を検討する。

アジャイルの組織的特徴は，開発チームが自己組織化されており，機能横断的となっていることである（西村・永瀬・吉羽，2013）。

ここでの自己組織化とは，状況に応じてミッションの実現するための選択を自分たちが決定し，行動できることである。また，機能横断的とは，外部に頼らず仕事ができるチーム，すなわち開発をやり遂げるために必要なあらゆるスキルを持つチームになっていることである（市谷，2019）。

自己組織化された組織では，能動的に，目的を実現するために自ら必要な情報を収集し，意思決定する必要がある。そのためには，問題が発生しても自分たちで解決したり，作業の指示を必要としたりしないようなスキルも持ち合わせた優秀なメンバーが必要となる（西村・永瀬・吉羽，2013）。

しかしながら，あらゆるスキルを備えたような人間はおらず，発生する問題を自ら解決できるような優秀なメンバーも常に揃えられるわけではない。アジャイルでは，メンバー一人ひとりが，すべてを完璧にこなせるような能力を持つことは必要とされない（西村・永瀬・吉羽，2013）。仮に不得意な分野であっても，チームを組んで対応することで，その方法を学んでいく。自身の作業範囲や役割を限定したり，チーム内でお互いに協力することをやめたりすることは禁じられている。

開発チームは，試行錯誤の「反復」活動を円滑に進めていかなければならず，重要なことは，一人ひとりの能力よりも，チームとしてさまざまな状況に対応できるかである。それぞれが持つスキルや経験，考え方，性格，得意不得意な分野などを理解し，開発を進めていくうえで，それらを考慮しながら進めていき，様々な問題を解決していくのである（西村・永瀬・吉羽，2013）。

特にソフトウェア開発は複雑で難しく（Ferguson, 1992），そのため，開発をスムーズに進めていくためにも，お互いを理解し，協力し合わなければ，少なくともユーザーが望むようなソフトウェアを作り上げることは不可能である（市谷，2019）。

アジャイルで求められていることは，自身の作業範囲を固定せず，全員で協力していくチームである。ソフトウェア開発は，単なるルーチンワークではなく，問題解決や試行錯誤を伴い，創意工夫や無から有を作り出す力が必要であり（Cusumano, 2004），そのような点を含む限り，難しい作業であり，お互いの経験や得意分野を持ち寄り，協力して助け合わなければならない。不得意な分野や作業であっても，協力し合って進めていくことで，学び，成長していく糧となるのである。

3.2 ふりかえり

次にアジャイル（スクラム）の大きな特徴の一つである，「ふりかえり」について検討する。

アジャイルでは，「ふりかえり」を実施し，カイゼンし続けることが重要となる。この「ふりかえり」とは，失敗を分析するような反省会だけではなく，改善のための集まりであり，開発プロセスや技術についてだけでなく，アジャイルのチーム自体もその対象となる。毎回の「反復」活動の終了のたびに話し合いの場を設け，うまくできたものや改善が必要なものを特定，整理し，次回の「反復」に向けた計画を立てることである。

アジャイルにとって重要なことは，失敗から学び，成長できるかである。「反復」活動を小さく，短く繰り返すことで，小さな失敗を繰り返し，多く学んでいくことになる。これは単なる試行錯誤ではなく，その失敗からふりかえ

り，学ぶことができる環境が必要となる。失敗が許されることで，そして，その失敗を糧にし，同じような失敗を繰り返さないようにチームメンバーが成長していくことで，最終的に高いパフォーマンスを発揮し，困難なミッションを達成していくのである（西村・永瀬・吉羽，2013）。

こうした理由のひとつとして，プロジェクトで生じる問題を発見できるのは，そのプロジェクトを実際に進めている現場の人間であり，その問題解決のための知恵を出せるのも現場の人間だからである（市谷，2019）。モノづくりにおいて，工場内の QC サークル活動やカイゼン活動を通じて人材育成がおこなわれ（齋藤，2009），そこから生まれた暗黙知などのナレッジが企業の中核となる資産となり，競争優位の源泉となっていったように，アジャイルも「反復」活動の中で「ふりかえり」を行い，失敗を繰り返しながらプロダクトをより良くしていく。

新しいことを体験し，学び，それを繰り返す努力をしながらチームは成長していくのであり，こうした組織的な学びの仕組み，つまり学習する組織が，アジャイル，特にスクラムの本質なのである（市谷，2019）。

4. ソフトウェア以外のアジャイル

現在のような VUCA（Volatility（変動性・不安定さ），Uncertainty（不確実性・不確定さ），Complexity（複雑性），Ambiguity（曖昧性・不明確さ））と呼ばれるような時代は，急速にグローバル化が進み，市場も急激な変化を遂げており，不確実性や不透明性が増した状況となっている。

アジャイルは，指揮命令系の管理手法とは根本的に異なる手法であるが，ソフトウェア開発の成功確率を劇的に高め，開発スピードと品質を改善する革命を起こしただけでなく，ラジオ番組の企画や，新しい機械の開発，戦闘機の生産，ワインの生産など幅広い業界や部門で広く取り入れられているという（Rigby, Sutherland, and Takeuchi, 2016）。

また，BMW や米国トヨタ自動車（TMS）といった自動車メーカーでも，組み込まれるソフトウェアだけではなく，自動車そのものの開発に取り入れられている（大西，2019）。

こうしたソフトウェア産業以外の分野でアジャイルのような方法が取り入れられている理由のひとつとして，俊敏性がある。組織やチーム内の状況に合わせて素早く対応を行うことである。しかし，ここまでアジャイルの特徴を確認してきたように，アジャイルの本質は，単に素早く物事に取り組むだけではなく，「反復」活動を通じてつくりあげたものに対するフィードバックと顧客との対話や交流を重視する点，さらに自己組織化や機能横断的な点，「ふりかえり」を重視する点である。特に，顧客からのフィードバックや市場の変化に応えるような開発中の設計の変更は，成功に結び付くようなものであれば，必ずしも悪いものではない。製品開発当初の計画に従うよりもユーザーの要望やビジネスの変化への対応を価値としているのである。

アジャイルは，特にネットビジネスを中心とする小さなスタートアップ企業やベンチャー企業で採用されることが多い。こうした企業は，創業して日が浅く，人数も少ないことが多いため，既存の企業で導入されているような階層型の組織に合わせた管理手法の導入が難しいことも多く，また，制度が固定化されていない。そのため，アジャイルに合わせて既存組織構造を破壊し，一から作り直す必要が無いため，アジャイルを導入する障壁が低いと考えられる。

アジャイルが向いている条件を表 2 に示す。

ソフトウェア産業以外の企業がアジャイルを導入するのは，イノベーションのためであり，決まりきった業務や作業工程にはあまり役立たないものの，こういった企業は非常に動きの激しい環境に置かれており，ただ新しい製品やサービスを開発するだけでなく，各部署の基本

的業務にもイノベーションを起こす必要があるためである（Rigby, Sutherland, and Takeuchi, 2016）。

さらに，近年はマーケティングや人事といった，プロダクトの開発以外の分野にもアジャイルは取り入れられ始めており，組織構造そのものをアジャイルに適応させようとする企業も現れ始めている。こうした企業では，制度の変更を行う際，法律のような緻密性や完璧性を優先するのではなく，6 ～ 7 割程度の簡素な形でビジネスの現場に導入し，ビジネスの実態に合わせて，運用しながら改良していくのである（松丘，2018）。

アジャイルを導入することでビジネスや市場に合わせて臨機応変に仕様を変更していくことが可能となる。また，最低限の仕様のみを決め，最速でビジネスを展開，ニーズに対応していくことも可能となり，こうした企業と相性がよいと考えられる。日常的にアジャイルが使われる環境を社内に構築することで，製品開発と基本的業務の両方においてイノベーションが生まれる速度が加速するのである。

ただし，アジャイルは万能薬ではない。例えば，アジャイル（スクラム）では，7 人前後の少人数のコミュニケーションを重視したチームを想定しており，ウォーターフォール・モデルのような数十人から数百人，さらには数千人規模といった大人数は想定していない。

また，表 2 にある通り，アジャイルは，解決すべき問題が複雑であったり，解決方法が不明であったり，要求される仕様が変更される可能性があるような環境を前提としている。そのうえで，作業がモジュール化でき，顧客との緊密なやりとりができ，よりクリエイティブなチームが重要視されるような状況が向いているのである。

アジャイルは，チームが最終的な結果を改善するために素早いサイクルで学習し，変化に迅速に適応できるようにするというものである。チームや企業の境界を超え，顧客と会話をし，共創の場の中で，イノベーションを作り出していく。それゆえに，アジャイルはもはや単なる

表 2　アジャイルの適用条件

項目	アジャイル向きの条件	アジャイルに不向きの条件
市場環境	顧客の要望と解決方法が頻繁に変わる。	市場環境は安定的で予測しやすい。
顧客の関与	緊密な共同作業と素早いフィードバックが可能。開発過程が進むにつれ，顧客は自ら望むものがはっきりしてくる。	要求仕様は当初から明確で，後になっても変化しない。 常に顧客との共同作業が可能なわけではない。
開発のタイプ	問題は複雑でソリューションは未知，または開発終了までの道筋もはっきりと定まっていない。 製品仕様は変更の可能性がある。創造的なブレイクスルーと市場投入までの時間が重視される。 部署横断的な共同作業が極めて重要。	似たような仕事を以前にしたことがあり，ソリューションは明白だと開発者は確信できる。 詳細な仕様と作業計画が自信を持って事前に予測でき，またその通りに進めなければならない。 各部門が部門内に閉じこもった作業を終えては次に回すことで問題が解決できる。
作業のモジュール性	インクリメンタル型開発が価値を持ち，顧客もそれを利用できる。 作業は部分ごとに分解でき，短期の作業を反復することで遂行できる。開発過程終盤になっての変更も処理可能。	顧客は製品が完成するまで，一部のパーツを使ったテストができない。 開発過程終盤の変更は大きな費用が掛かる，もしくは不可能。
一時的なミスのもたらす影響	そこから貴重な学びが得られる。	それが致命的な打撃となりかねない。

出所：Rigby, Sutherland & Takeuchi（2016）より。

ソフトウェアの開発手法ではなく，学習する組織活動として捉えることが可能であり，ビジネスを作る人々の活動の根底となり得るのである。

5. おわりに

本研究では，アジャイルの本質を確認するとともに，ソフトウェア開発以外の分野に適用するための要因を検討した。アジャイルの本質は，顧客との共創を前提とした「反復」活動を通じた組織的な学習である。顧客にとってより良いものをつくるためには，完成させるのではなく，「反復」を通じてフィードバックをもらい，毎回の成功や失敗から学習し，カイゼンしていくのである。こうした学習する組織を作り上げるには，マネジャーといった管理者の指揮の下ではなく，能動的に動ける自己組織化，そして機能横断的なチームになることが重要であり，こうした取り組みの要素はソフトウェア分野に限定されるものではなく，自動車などの他のプロダクトやマーケティングや人事などにも適用されているのである。

一方で，ソフトウェア開発における方法をそのままサービスや組織，人事といった他の分野に適用することは当然ながら難しいと考えられ，こうしたアジャイルの取り組みの要素を具体的にどのように当てはめているのか，その実態を詳しく調査していく必要があろう。また，日本に適したアジャイルの導入といった視点もあり得るであろう。

注

1）平鍋・野中（2013）によると，野中郁次郎と竹内弘高による 1986 年の論文“The New Product Development Game”において，日本の製造業の新製品開発からヒントを得て「スクラム」と名付けられ，その後，1990 年年代前半に，Jeff Sutherland，John Scumniotales，Jeff McKenna によって，ソフトウェア開発手法として開発されたという。
また，Takeuchi and Nonaka（1986）によると，新製品開発のプロセスを 3 つのタイプに分類しており，TypeA は工程が連続的にリレーしている方式，TypeB はプロセスの前後が重複した方式，そして TypeC はプロセスがよりオーバーラップした方式としている。このうち TypeC についてラグビーをもとに「Moving the Scrum Downfield」と名付けている。

参考文献

Bean, Michael (2005) “The Pitfalls of Outsourcing Programmers: Why Some Software Companies Confuse the Box with the Chocolates,” *The Best Software Writing I: Selected and Introduced by Joel Spolsky*, Edited by Joel Spolsky, Apress LP., pp.9-15.（青木　靖訳「プログラマのアウトソーシングの落とし穴－なぜソフトウェア会社はチョコレートと箱を取り違えるのか」『BEST SOFTWARE WRITING』翔泳社，2008 年）

Cusumano, Michael A. (1991) *Japan's software factories: a challenge to U.S. management*, New York; Tokyo: Oxford University Press.（富沢宏之・藤井留美訳『日本のソフトウェア戦略：アメリカ式経営への挑戦』三田出版会，1993 年）

Cusumano, Michael A. (2004) *The business of software: what every manager, programmer and entrepreneur must know to thrive and survive in good times and bad*, Simon and Schuster.（サイコムインターナショナル訳『ソフトウエア企業の競争戦略』ダイヤモンド社，2004 年）

Ferguson, Eugene S. (1992) *Engineering and the Mind's Eye*, The MIT Press.（藤原良樹・砂田久吉訳『技術屋の心眼』平凡社，1995 年，2009 年）

Glass, Robert L. (2006) *Software creativity 2.0*, Developer. * Books.（高嶋優子・徳弘太郎・森田　創訳『ソフトウエア・クリエイティビティ：ソフトウエア開発に創造性はなぜ必要か』日経 BP 社，2009 年）

Rigby, Darrell K., Sutherland, Jeff, and Takeuchi, Hirotaka (2016) “Embracing Agile,” *Harvard Business Review*, MAY 2016 ISSUE, Harvard Business School Publishing.（倉田幸信訳「アジャイル開発を経営に活かす 6 つの原則 臨機応変のマネジメントで生産性を劇的に高める」『ハーバード・ビジネス・レビュー』2016 年 9 月号，ダイヤモンド社，2016 年）

Royce, Winston W. (1970) “Managing the Development of Large Software Systems,” Proceedings of IEEE WESCON.

Takeuchi, Hirotaka and Nonaka, Ikujiro (1986) “The New New Product Development Game,” *Har-*

vard Business Review, Harvard Business Publishing.
市谷聡啓（2019）『正しいものを正しくつくる―プロダクトをつくるとはどういうことなのか，あるいはアジャイルのその先について―』BNN 新社.
大西孝弘（2019）「BMW，アジャイル開発で目指すスピード経営」『日経ビジネス』2019 年 2 月 25 日号，日経 BP，pp.70-74.
小椋俊秀（2013）「ウォーターフォールモデルの起源に関する考察―ウォーターフォールに関する誤解を解く」『商学討究』Vol.64, No.1, pp.105-135, 小樽商科大.
大日方真（1971）「プログラム開発の工程管理（I）」『情報処理』Vol.12, No.10, pp.627-636, 一般社団法人情報処理学会.
齋藤　毅（2009）「生産システム論の再構成：労働研究と経営研究の統合に向けての試論」『評論・社会科学』Vol.88, pp.145-191, 同志社大学.
妹尾　大（2001）「ソフトウェア開発の新潮流―状況論的リーダーシップの胎動」『組織科学』Vol.35, pp.65-80, 組織学会.
高橋信弘（2010）「日本のソフトウェア産業の国際競争力に関する一考察：国際競争力欠如の諸要因とその因果関係」『經營研究』Vol.60, No.4, pp.151-167, 大阪市立大学.
独立行政法人情報処理推進機構（2011）「グローバル化を支える IT 人材確保・育成施策に関する調査 調査報告書」平成 23 年 3 月，https://www.ipa.go.jp/files/000010609.pdf.
西村直人・永瀬美穂・吉羽龍太郎（2013）『SCRUM BOOT CAMP THE BOOK』翔泳社.
平井直樹（2018）「ソフトウェア開発プロセスにおける分業構造と知識労働―日本の受託ソフトウェア開発の組織問題―」博士論文，立教大学大学院ビジネスデザイン研究科.
平鍋健児・野中郁次郎（2013）『アジャイル開発とスクラム』翔泳社.
松丘啓司（2018）「アジャイルな人事変革の必要性―ビジネスのイノベーションを加速するために―」『経営センサー』9 月号，No.205, 東レ経営研究所.

資　料

James, Michael. and Walter, Luke. (2010) “Scrum Reference Card” http://scrumreferencecard.com/（スクラムリファレンスカード，http://scrumreferencecard.com/ScrumReferenceCard_v1_3_1-jp.pdf）（2019 年 9 月 4 日現在）
日経ビジネス編（2019）「BMW（独自動車大手）アンチ巨人のスピード経営」『日経ビジネス』2019 年 2 月 25 日号，日経 BP 社.