ソフトウェア開発におけるアジャイルに向けた共創の課題
― 顧客の構造問題の観点から ―

平井 直樹 [1]

Problem of Co-Creation for Agile in Software Development: From the Perspective of Customer Structural Issues

Naoki Hirai

1.背景と研究目的

ソフトウェア開発では、これまでの契約のもとに計画と遂行、その管理を中心とするウォーターフォール・モデルから、決められた計画よりも変化へと柔軟に対応し顧客価値を中心とするアジャイルへと切り替わろうとしている。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、ソフトウェアの開発にもスピードが求められるようになり、特にアメリカを中心にアジャイルと呼ばれる手法が優先的に採用されている（West and Grant, 2010）。

このアジャイルは、ソフトウェア開発のすべての仕様が確定されてからではなく、ある程度決まったものから開発を行い、1 週間や 2 週間などの一定の期間内で一つの機能を開発し、リリースすることを繰り返す。後述するウォーターフォール・モデルと比較してすぐにソフトウェアを利用できるようになるうえ、そうしたある程度完成したソフトウェアが利用できるため、顧客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。

アジャイルは、短いサイクルを回す試行錯誤型の開発プロセスであるが、その本質は、顧客との共創であり、顧客との綿密なコミュニケーションとフィードバックが求められる。日本では 1950 年代から続くウォーターフォール・モデルが主流であり、アジャイルにはほとんど移行していない（妹尾, 2001; 高橋, 2010）。この理由として、アジャイルに沿った開発の難しさだけでなく、日本特有のソフトウェア開発の外注や下請けの問題があることが想定され、そこでは共創の取り組みができていないと考えられる。

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、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であり、計画の立案やその実行と管理が行いやすいといった利点がある（新井, 2016）。しかし、予め決められた計画と目的に従うという制約のため、顧客の要望が途中で変わっても対応することができない。ビジネス

[1] 昭和女子大学 現代ビジネス研究所 研究員／立教大学大学院ビジネスデザイン研究科 助教

環境の変化と速さに対応し、顧客の要望を反映させた真に価値あるソフトウェアを作り上げるためには、顧客自身の協力が必要となる。

こうした共創については、顧客価値と結び付けられマーケティング分野で研究が多い。また、コミュニケーションを行い相互に理解し働きかけ合い、心理的に刺激する状況の枠組みとしての「場」の研究もある（伊丹, 2005）。しかし、ソフトウェア開発に関する研究は、工学分野が多く、経営学や社会学の視点からそうした顧客を中心に置いた共創に関する研究は少ない。

本研究では、アジャイル、ウォーターフォール・モデルのそれぞれの特徴を確認するとともに、内製と外注、下請け構造といった顧客の構造問題の観点から日本のソフトウェア開発が何故アジャイルにおける共創ができていないのか、その課題を検討する。

2.アジャイルとウォーターフォール・モデルによる開発方法の比較

2.1 アジャイルの開発方法の特徴

アジャイルは、1990 年代に登場したスクラム（Scrum）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、後述する従来のソフトウェア開発手法であ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; Coplien and Harrison, 2004; James and Walter, 2010; 設楽・中佐藤編, 2012）。

アジャイルとウォーターフォール・モデルの開発手法の比較を表 1 に示す。

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る（永里, 2018）。また、ステークホルダーからの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といった共創を重視している（平井, 2019）。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）、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、その開発期間も 1 週間から 1 か月程度の短い間隔で反復して行い、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。反復作業も要件の中で最優先の高いものからできるだけ早く作成し、少しでもできあがったソフトウェアをユーザーに利用、確認してもらうことで、早期にフィードバックを得て、再度反復作業に戻るのである。

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、戦闘機の製造、ワインの生産のほか（Rigby, Sutherland, and Takeuchi, 2016）、BMW や米国トヨタ自動車といった自動車メーカーの自動車そのものの開発（日経ビジネス編, 2019）、GE や IBM（Cappelli and Tavis, 2018）

など幅広い業界や部門で広く取り入れられている。

表 1 アジャイルとウォーターフォール・モデルの開発手法の比較

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

出典：Glass（2006），平鍋・野中（2013）、平井（2018）をもとに作成

2.2 ウォーターフォール・モデルの開発方法の特徴

ウォーターフォール・モデルは、製造業をモデルとし、原則として作業の順番が決まっており、上流工程を重視し仕様変更を防ぎ、納期や費用という観点で効率的に管理しようとするものである。厳格な手順を守ることを重視し、プロジェクトの開始時点で顧客からの要求とそれに伴う要件や仕様が既に確定しており、プロジェクトが完了するまで変更されないことが前提となっている（設楽・中佐藤, 2012）。つまり、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。

ウォーターフォール・モデルでは、顧客からの要望を取り入れて、仕様を明確にして設計に落とし込む上流工程と、その仕様や設計を元にプログラムを作り上げてテストを行う下流工程とに大きく分離され、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。この仕様変更を極力防ぐ設計手法は、開発の初期段階で問題を発見することで、後工程での設計変更を減少させ、前工

程への手戻りを防ぎ、開発コストの増加や開発期間の長期化を防ぐことを目的としている。そのため、特に上流に位置する基本設計や概念設計に重きが置かれている（竹村, 2001）。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（Royce, 1970; 小椋, 2013）。

ウォーターフォール・モデルは、計画に沿って進めていくため、スケジュールの立案や開発プロジェクトの全体を把握しやすく、資源配分、進捗状況の理解が容易となる。一方で、工程の遡りが発生しないように必ず前工程が完了する前提のため、要件定義、設計や製造といったそれぞれの工程ですべての作業が終了しなければ次工程へ着手できず、開発スケジュールが全体的に長くなってしまうと同時に、すべての工程が完了し、完成しなければリリースができないという問題も存在する。

2.3 アジャイルに求められる条件

表 1 でそれぞれの開発手法を比較したが、アジャイルはこうした開発手法の前提のもと、チーム内の組織的な学びの仕組みや顧客との共創の特徴が存在する。

アジャイルの適用条件を表 2 に示す。

本研究では、このうち、顧客との共創に関わる項目として、「顧客の意思決定」および「顧客の関与とフィードバック」「ステークホルダー内の関係」に特に注目する。ここでの顧客とは、ソフトウェア開発企業やその開発チームが直接やりとりを行う企業担当者のことであり、主に情報システム部門を指す。また、ステークホルダーとは、そうした顧客の関係者であり、出来上がったソフトウェアを利用する部門や部署のことであり、後述する事例では、統合する企業の各部門や製品を実際に管理している部署などを指す。

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。アジャイルは、開発チームが状況に応じてミッションを実現するための選択を自分たちが決定し、行動できる自己組織化がされており、また、外部に頼らず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として機能横断的となっている（西村・永瀬・吉羽, 2013; 松丘, 2018; 市谷, 2019）。

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る（Cusumano, 2004）。アジャイルにおける開発チームは、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どうかである。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）。

アジャイルにおける試行錯誤の「反復」活動は、単なるトライ&エラーではなく、また、失敗を分析するだけのような反省会でもなく、改善のための「ふりかえり」が重視されている。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や難易度を達成していく（西村・永瀬・吉羽, 2013)、こうした組織的な学びの仕組み、つまり学習する組織活動がアジャイルの本質である（平井, 2019)。

表 2　アジャイルの適用条件

項目	アジャイルが適用できる要因	アジャイルが適用できない要因
市場環境	顧客の要望と解決方法が頻繁に変わる（変化が多い)。	市場環境は安定的で予測しやすい（変化が少ない)。
予算や納期	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。	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。
問題解決の見通し	問題は複雑で解決方法は未知、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。	似たような仕事を以前にしたことがあり、解決方法は明白だと開発者は確信できる。
顧客の意思決定	要求仕様はあるが、開発過程が進むにつれ、顧客の望むものがよりはっきりする（目的が変わっていく)。	顧客からの要求仕様は当初から明確となっている。また、後になっても変化しない。
顧客の関与とフィードバック	緊密な共同作業と素早いフィードバックが可能。	常に顧客との共同作業が可能なわけではない。
ステークホルダー内の関係	部署横断的な共同作業が極めて重要。	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	開発規模が小さいため、大きな打撃にはならない。	開発規模が大きいため、致命的な打撃となりかねない。
開発成果からの学び	次の反復開発に学びを生かすことができる（成功、失敗に関わらず成長していくことが前提)。	次の開発まで生かせない（開発工程の後戻りができないため)。

出典：Rigby, Sutherland, and Takeuchi（2016)、平井（2019）をもとに作成。

また、アジャイルは、顧客との綿密なコミュニケーションとフィードバックを基にする共創が求められる。アジャイルかウォーターフォール・モデルかの選択は、開発するソフトウ

ェアの規模や納期のほか、どのようなリリースや運用をしたいのかなど、顧客側の事情やその意思決定による影響が大きい。しかし、日本では、ソフトウェアの開発を行う際、自社にシステムエンジニアやプログラマーといった人材が不足していたり、企業の開発部門の力が弱いとされる（藤田, 2013）。そのため、内製化が難しく、子会社などの下請けや他のソフトウェア開発企業へ外注が行われているケースが多い。

しかし、ソフトウェア開発は複雑で難しく（Ferguson, 1992）、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、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できない（市谷, 2019）。表 2 で挙げたようにアジャイルを採用するのであれば、顧客は外注先のソフトウェア開発企業と綿密な連携を行うと同時に、顧客自身もソフトウェアの開発に積極的に関わるという考えや体制に切り替えなければならず、そこに日本のソフトウェア開発の構造的な問題があると考えられる。

3.日本のソフトウェア開発の構造的な問題

次に日本のソフトウェア開発の特徴として顧客の構造問題について確認する。日本でアジャイルのような顧客との共創が難しい理由として、内製と外注、下請け構造の問題が考えられる。

3.1 ソフトウェア開発の内製と外注

ソフトウェアの開発に限らず多くの産業では、その複雑な作業工程を分割、分業することで成り立ってきた。複雑なシステムで構成されている作業は、組織の階層構造や分業制度によって負荷を削減し、複雑な作業を細かくモジュール化し分業することで、個人での処理や外部の企業に外注することも可能となり、この方法はウォーターフォール・モデルと相性が良い（平井, 2018）。

上流に位置する工程では、高度なスキルを持ち、業務に精通した熟練技術者が設計図を作成する。一方で、下流に位置する工程では、コストが高いベテラン社員より、スキルや経験が浅いもののコストが安い新入社員などの技術者や、自社の社員よりコストがかからない外注先の企業に割り振る方法が、ウォーターフォール・モデルに準拠した日本の標準的なソフトウェアの開発方法である（平井, 2015）。

製品開発のように、その内容や成果の見極めが難しく不確実性が高いものの場合、適切な相手企業を見つけて取引や行動を監視するよりも、その業務を内部化したほうが効率的となる（武石, 1999）。近年のソフトウェア開発は、事業のスピードが速くなるとともに、複雑化し、不確実性も高くなっているため、内製した方が効率的であるともいえる。

一方で、ソフトウェア開発企業と市場規模の整備が進むことで取引コストが低下し、さらに多様で高度なソフトウェアへのニーズが増大し、そのためのマネジメントコストが増大することで、ソフトウェア開発も内製から外注に移行していった（今井他, 1989）。

日本のソフトウェア開発が、このような外注に頼る理由の多くは人材確保や開発コストの削減である。企業が特定業務を専門の外部企業に任せることで、優れた技術の利用や時間の短縮、費用の節約が期待できる。特に、分業により自社はコアとなる業務の強化に集中できることが利点とされ、ソフトウェア開発で外注が行われる要因の一つとなっている。

こうした利点の一方、外注に任せることで、自分たちでソフトウェアを作り上げるという能動的な姿勢は少なくなり、開発者と一緒に作り上げるというよりも、納品してもらうという考え方に陥りやすく、顧客自らが本当に望むソフトウェアの開発は難しくなる。試行錯誤を経て魅力的な機能を持つようなソフトウェアを開発する方法としては最良ではないのである（Cusumano, 2004）。

3.2 下請け構造

ソフトウェア開発では、要件定義や設計を分離し、特に下流工程に位置するプログラミング作成やテストを下請けの子会社やインドなどの海外の企業に外注することで（Cusumano, 2004）、自社で不足する技術者の確保やコストの削減を図ってきた。

日本のソフトウェア開発では、一企業に限らず、複数の企業と開発を行ったり、下請け企業として何層にもわたるような構造をもつケースも存在する（峰滝・元橋, 2007; 平井, 2012）。特にこうした下請けが利用される原因は、顧客の社内にソフトウェアを開発できる人材がいないだけでなく、ソフトウェア開発企業の中でも常に必要となる人材がいるわけではないためである。IT 産業では、技術革新が早く、その技術分野も多岐に渡り、さらに顧客の業種や業務の繁閑の差は大きいため、自社内であらゆるケースを想定した人材の確保は難しく、現実的ではない（Cappelli, 1999; 若林, 2008）。そのため、余剰な人材は社内に抱え込むのではなく、下請け企業を利用することで要員調達を柔軟にし、効率的に業務に組み込めるようにしてきた。

一方で、ソフトウェアを含む IT 産業全般で先行しているアメリカでは、数多くのプロフェッショナルサービス企業が種々のソフトウェアを開発、販売しており、日本のような外注を中心とした下請け構造を形成していないとされる。北米では日本より産業の細分化が進んでおり、4 人以下のスモールサイズの企業が多く、カナダのエドモントンで行なわれた調査では、下請けビジネスの存在自体が確認されていない（高木, 2007）。

しかしながら、日本の雇用制度では、正社員を解雇することは容易ではない。ソフトウェア開発は、労働集約型の産業であることから設備投資とは異なり、社員の稼働率を低下させることで生産調整をするような方法が取りづらい。その結果、親企業、グループ企業間で技術者を融通し合うケースが多い。

こうした下請け関係は、一方的な命令などのやりとりになる場合も多い。下請け制度による分業システムは、バッファー的に利用される存在として捉えられ、顧客からの発注が一方的に打ち切られやすいほか、親事業者の優越的地位の濫用によって、下請け事業者が取引において不利益を被ることが挙げられ、また取引が複雑になることからコミュニケーション

ロスによる業務の非効率性、責任の所在の不明確化などが懸念される。

このようにソフトウェア産業の下請け構造は、顧客と開発者、または、企業間同士の密接なコミュニケーションやフィードバックをもった開発が難しくなる要因の一つとなっている。ここに日本のソフトウェア開発、特にウォーターフォール・モデルに準拠した開発の弱点があるといえる。

以上のように、人材不足やコスト削減から内製化ができないことによる外注の促進と下請け構造が日本のソフトウェア開発の特徴である。こうした構造は、顧客と開発企業との関係に、隔たりを作るだけでなく、受発注関係のもと、顧客は受け身の姿勢になりやすくなる。さらに、顧客取引が複雑になることで、ステークホルダーとの調整も難しくなる。このため、アジャイルのような綿密なコミュニケーションとフィードバックを基にする共創という考え方自体を難しくしていると考えられる。

4.事例

実際に企業の現場では、顧客との共創の取り組みはどのように行われているのであろうか。本研究では、合併・統合、ステークホルダーとの調整の 2 点の事例を挙げ、表 2 で示したアジャイルの適用条件を分析の枠組みとし、日本のソフトウェア開発の共創の問題を明らかにする。なお、事例については、守秘義務や顧客企業との関係のため、対象の企業や開発プロジェクトの内容が特定されないよう処理するとともに、対象企業において調査、開示可能な範囲での分析としている。

4.1 企業の合併・統合の事例

金融系システム開発で、金融機関同士の合併・統合の事例となる。時期は 2010 年頃の金融機関同士の大規模な合併であり、経営統合に続き、システム統合が行われ、筆者もこのような統合作業に従事していた経験がある。

この事例のアジャイルに関わる要因について、表 3 に示す。

企業の合併・統合は、必然的に開発規模も大きくなる傾向にある。規模の拡大を追及したり、お互いの事業領域や製品の弱点を補完しあうなどグローバルへの対応も含め、企業の合併・統合は今後も多く起こることが想定される。

金融機関でも合併・統合は多く発生しており、それに伴い、内部システムの統合や改修も何度も行われている。しかし、2002 年 4 月 1 日のみずほ銀行のシステム統合の失敗により、統合における障害の影響の大きさが明らかとなった。この時は、現金自動預入払出機（ATM）の障害が発生した。そのうえ、口座振替の遅延や二重引き落としまでも発生し、復旧に 1 か月以上の期間を要した。こうした事態を受け、金融庁は、金融機関同士の合併・統合の際には、より厳密な監査を行うようになっている（日本銀行金融機構局, 2009; 金融庁 2019a,

2019b）[2]。企業側もソフトウェア開発の進行段階において、こうした監査に対応できるよう、設計書などの文書や、開発手順の順守、検証の結果などさまざまな証跡を準備する必要がでてきた。

しかし、アジャイルのような日々変化する開発方法は、こうした大型の統合作業では所構わず変更を行うと収集がつかなくなってしまい、進捗の段階を経た監査に対応できない。そのため、予め計画を立ててウォーターフォール・モデルに基づいて開発を行う必要が生じる。そこで企業は工程ごとに紙に印刷された文書による確認を行い、その結果を保管していく。そうした文書は、ダンボールに梱包され倉庫に山積みとなっていたが、実際のところ監査で全てが読まれるわけではなく、問題が発生した際の担保として保管されていた。

表 3 合併・統合の事例

項目	状況
市場環境	市場環境は、大きく変わってきているが、ある程度予測は可能。
予算や納期	納期となるシステムの統合日は多くの社内システムの他、企業全体のビジネスなどの調整を経て決められており、変更することはできない。
問題解決の見通し	企業同士の合併・統合のため、解決すべき問題は複雑で多い。解決の見通しはある程度立っていた。
顧客の意思決定	要求仕様は当初から明確になっている。しかし、安全なシステム統合が最大の目的であり、より良いソフトウェアの開発といった余地は非常に少ない。
顧客の関与とフィードバック	顧客の関与は多いものの、顧客自身が合併・統合するそれぞれの企業内の担当者との擦り合わせも多く、確認に手間や時間がかかりフィードバックをもらいにくい。
ステークホルダー内の関係	関連部署・部門などのステークホルダーとの調整だけでなく、統合する企業間の調整も必要となり、共同作業は非常に難しい。
作業のモジュール性	統合に伴うシステムは、一システムだけが完成しても、動かすことはできない。全システムが完成して初めて動かすことができるため、歩調を合わせて進める。
一時的なミスのもたらす影響	開発規模も多く、さらにミスは他のシステムに影響してしまうため、致命的な打撃となる。1 つのシステムの障害が、他のシステムへも波及してしまい、スケジュールが逼迫してしまった。
開発成果からの学び	大きなプロジェクトであり成長はできるが、こうした大きなプロジェクトは毎回関われるわけではなく、次回に生かせるかどうかは不明といえる。

出典：Rigby, Sutherland, and Takeuchi（2016）、平井（2019）をもとに筆者作成

[2]「プロジェクトの規模・特性やリスク等を踏まえて、工程ごとの作業進捗、品質、工程開始・終了判定などに問題ないかを監査するといったプロジェクト監査を行うことが必要」金融庁（2019b）p.6 とされている。

この事例は、企業の合併のため関連部署・部門などのステークホルダーとの調整や共同作業が困難という構造的な問題があり、アジャイルには不向きな、ウォーターフォールを採用せざるを得ないプロジェクトであった。やや政治的な理由もあるが、企業が合併・統合のような大型化、複雑化する状況において、絶対に失敗が許されないようなソフトウェア開発では、アジャイルのような失敗を糧に試行錯誤を繰り返す開発方法よりは、ウォーターフォール・モデルのような事前に計画を立てて目的に向かって確実に開発を進める方法が採用されるのは合理的であると考えられる。

また、顧客の目的は、より良いソフトウェアを開発し顧客価値を実現するよりも、安全なシステム統合となっていた。合併・統合のため、要求仕様も当初から明確になっているが、その内容自体も新しいものを作るというよりは、いかに安全に統合が果たせるかであり、そこに改善の余地は非常に少ない。こうしたプロジェクトでは、顧客の関与は多いものの、それは目的のものが予定通りできているかどうかの管理の意味合いが強い。合併・統合する双方の顧客同士の擦り合わせや調整が多く、ソフトウェア開発を共同で行うということは少ない。このため、アジャイルの特徴であるフィードバックと振り返りによる試行錯誤の反復活動は行われておらず、こうした状況では共創自体が難しかったと考えられる。

4.2 多数のステークホルダーとの共創の事例

顧客の社内に開発ができる人材がいないため、ソフトウェア開発を外注した一般的な開発の事例である。2019 年春頃に行われた開発で、一般向け製品販売の Web サイトとそのメンテナンスのシステムのリニューアルを目的としている。しかし、対象となる製品が多数あることで、いくつもの部署に跨ってステークホルダーが存在していた。

この事例のアジャイルに関わる要因について、表 4 に示す。

多くの企業では、社内に開発要員を確保していることは少なく、ソフトウェアやシステムを管理する情報システム部が下請けや外注で開発を行うことが多い。情報システム部は仲介役となり、他部署などのステークホルダーから依頼を受けて開発を行い、その内容も社内システムの開発から、Web サイトの開発、業務のデジタル化など多岐にわたる。

事例では、顧客である情報システム部の関与は多いものの、実際に製品を管理している営業企画部などの各部署は、開発に関わるような共同作業という形ではなく、ほぼできあがったものを実際に利用してみることでフィードバックを行うというスタイルであったため、素早いフィードバックがもらえるわけではなかった。

また、要求仕様は決められていたが、実際の Web 画面のレビューを行ううちに、顧客の要望は変わっていった。しかし、予算と納期は決まっていたため、顧客の本当に望んでいたものまでは十分に反映できなかった。反映できなかったものについては、次回の開発プロジェクトへと持ち越しとなった。

こうした開発では、顧客だけでなく関係部署などステークホルダーとのミーティングも行われるが、必ずしも協力的な相手ばかりではない。また、そうしたミーティングでは事前

の要望が反映されているかの確認が中心となることが多く、書類上での確認のほか、その時点ではソフトウェアは未完成のためイメージのみを提示しての確認などが行われた。特に、ステークホルダーすべてに直接確認できるとは限らず、情報システム部を介しての確認となることも多かったため、フィードバックなど意思の疎通が難しい状況であった。

表 4 ステークホルダーとの調整事例

項目	状況
市場環境	他社のシステムなどと比較されることで、要望が追加されることもある。
予算や納期	早く一般顧客に利用してもらいたいがため、早めの納期が求められていたが、予算は決められており、新たな要望が追加されても反映できる余地は少ない。
問題解決の見通し	複雑な問題はなく、解決すべき問題も明らかとなっている。
顧客の意思決定	要求仕様は決められている。しかし、実際に開発された Web 画面（ソフトウェア）を確認することで、顧客の望むものが変わっていった。
顧客の関与とフィードバック	直接の顧客であるシステム部の関与は多い。ただし、各部署の担当者は途中ではなく出来上がってからチェックをするため、素早いフィードバックがもらえない。
ステークホルダー内の関係	部署間の調整が必要となるほか、そうしたステークホルダーの一部の意見により、終盤になって仕様が覆るようなこともあり、あまり共同作業はできておらず、開発スケジュールの逼迫の原因となった。
作業のモジュール性	大きなシステムではないため、部分ごとに開発が行われた。製品についてもがデータベースを種類ごとに修正するだけであった。
一時的なミスのもたらす影響	小さめのプロジェクトであったため、問題が発生してもある程度すぐにリカバリーが可能。
開発成果からの学び	何度も作り直しているわけではなく、当プロジェクト中の成功や失敗からの学びをすぐに生かすことはできなかった。

出典：Rigby, Sutherland, and Takeuchi（2016）、平井（2019）をもとに筆者作成

この事例では、ステークホルダーとの調整が不足しており、さらに一部は積極的には協力してくれず、共に作り上げているという観点を持っていなかった。自社の人材不足から開発を外注に任せてしまい、出来上がって納品されたものを確認するだけという受け身のスタイルであった。それゆえ、顧客は要望をきちんと開発企業側に伝えることができず、終盤になってからの変更が発生し、開発に混乱をもたらしてしまったのである。顧客が直接に開発に関われないような方法を採用する限り、またそうした受け身の顧客体制である限り、そこに共創は存在せず、アジャイルのような考え方の導入は難しいといえる。

5.まとめと今後の課題

本研究では、アジャイルとウォーターフォール・モデルの特徴を確認するとともに、アジャイルの特徴のひとつである顧客との共創に焦点を当て、何故日本のソフトウェア開発では、そうした共創が行えないのか、検討を行った。

先行研究や事例より、開発企業と顧客の間でコミュニケーションやフィードバックが貰えにくくなる共創を妨げる構造的な問題が多くあることが明らかとなった。日本のソフトウェア開発は、顧客の人材不足や開発コストの問題から外注や下請けに頼ることが多い。失敗が許されないような開発や、予算や計画ありきで開発を進めざるを得ない状況もあるが、そうしたケースでは、ソフトウェアは開発され、納品されるものという受け身の体制となりやすく、必然的に顧客を含むステークホルダーの関与が少なくなりやすい。そのため、アジャイルのような目的が変化することを許容し、顧客にとって価値のあるより良いソフトウェアを作ろうとするような開発や、顧客を最初から最後まで巻き込み一緒に作り上げる共創という考え方が乏しい。ソフトウェア開発は複雑で難しく、お互いを理解し、協力し合わなければ、顧客が望むものは作ることはできない。アジャイルのような、ステークホルダーを含む顧客との綿密なコミュニケーションとフィードバックを主体とした共創が重要であり、日本のソフトウェア開発に不足しているのである。

今後の研究課題として、本研究では事例から開発における共創の問題を提示したが、アジャイルにおける共創が実際に企業内でどのように行われており、どのような効果が発揮されているのかについてまでは本研究の範囲外のため、踏み込んでいない。また、外注や下請けの問題を提示したが、先行研究で示したように現実的には日本では内製が難しく、また内製にしたところで従来通りの開発手法をとるのであれば共創ができる可能性は少ない。一方で、ソフトウェア開発以外にもアジャイルが近年利用されており、そういった企業ではどのような共創が行われているのか、明らかにしていく必要があろう。

また、本研究では、ソフトウェア開発における共創の定義を十分に分析できていない。共創については、マーケティングの分野で顧客価値と絡めて多くの知見が蓄積されている。ソフトウェア開発では、コミュニケーションやフィードバックがアジャイルにおける顧客との共創に位置されているが、関連分野の先行研究、および具体的な作業内容から共創の定義と仕組みを明らかにする必要があろう。

参考文献

新井雄一朗（2016）「定性的なソフトウェアプロジェクトデータに基づくプロダクト品質予測に関する研究」『法政大学大学院理工学・工学研究科紀要』第 57 巻, 法政大学大学院理工学・工学研究科, 1-6 頁。

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―』株式会社 BNN 新社。

伊丹敬之（2005）『場の論理とマネジメント』東洋経済新報社。
今井賢一・安藤博・白井豊・辻淳二・久保宏志・玉置彰宏・浜田淳司（1989）『ソフトウェア進化論』NTT 出版。
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』第 64 巻, 第 1 号, 小樽商科大, 105-135 頁。
金融庁（2019a）「システム統合リスク管理態勢に関する考え方・着眼点（詳細編）」平成 31 年 3 月。
金融庁（2019b）「システム統合・更改に関するモニタリングレポート」令和元年 6 月。
設楽秀輔・中佐藤麻記子編, 長瀬嘉秀監修（2012）『アジャイル概論』東京電機大学出版局。
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』第 35 巻, 組織学会, 65-80 頁。
高木義和（2007）「日本と北米における情報サービス産業の構造比較」『新潟国際情報大学情報文化学部紀要』第 10 巻, 新潟国際情報大学, 119-130 頁。
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』第 60 巻, 第 4 号, 大阪市立大学, 151-167 頁。
武石彰（1999）「製品開発の戦略的アウトソーシング」『研究技術計画』第 14 巻, 第 2 号, 研究・イノベーション学会, 108-114 頁。
竹村正明（2001）「現代的な製品開発論の展開」『組織科学』第 35 巻, 第 2 号, 組織学会, 4-15 頁。
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。
永里賢治（2018）「日系企業の海外進出マネジメント:－中国におけるものづくりと戦略的提携－」『国際 P2M 学会研究発表大会予稿集』2018.Autumn(0), 一般社団法人国際 P2M 学会, 185-196 頁。
西村直人・永瀬美穂・吉羽龍太郎（2013）『SCRUM BOOT CAMP THE BOOK』翔泳社。
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, 70-74 頁。
日本銀行金融機構局（2009）「金融機関におけるシステム共同化の現状と課題―地域銀行 108 行へのアンケート調査結果から―」。
平井直樹（2012）「中小ソフトウェア企業における下請け構造の実態と問題点」『経営会計研究』第 16 号, 日本経営会計学会, 41-54 頁。
平井直樹（2015）「わが国の製造業をめぐる組織戦略としてのソフトウェア開発方法の課題 ―受託ソフトウェア産業のモデル分析―」『経営会計研究』第 19 巻, 第 2 号, 日本経営会計学会, 121-139 頁。
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。

平井直樹（2019）「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―」『立教 DBA ジャーナル』第 10 号, 立教大学大学院ビジネスデザイン研究科, 35-44 頁。

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。

藤田哲雄（2013）「わが国の電機産業の再生に向けて」『Japan Research Institute review』第 6 巻, 第 7 号, 日本総合研究所, 57-81 頁。

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』2018 年 9 月号, 第 205 号, 東レ経営研究所, 54-58 頁。

峰滝和典・元橋一之（2007）「日本のソフトウェア産業の業界構造と生産性に関する実証分析」, RIETI DISCUSSION PAPER SERIES 07-J-018, 独立行政法人 経済産業研究所。

若林直樹（2008）「専門能力を持つ人的資源の開発における企業間協力の考察－欧州製薬団体連合会の合同開発職研修の事例分析」『経済論叢』第 181 巻, 第 1 号, 京都大学経済学会, 104-122 頁。

Cappelli, Peter.（1999）, *The new deal at work: managing the market-driven workforce*, Harvard Business School Press.（若山由美（2001）『雇用の未来』日本経済新聞社。）

Cappelli, Peter. and Tavis, Anna.（2018）,"HR Goes Agile", *Harvard Business Review*, March-April 2018, Harvard Business School Publishing.（有賀裕子訳（2018）「採用、評価から育成まで アジャイル化する人事」『ハーバード・ビジネス・レビュー』2018 年 7 月号, 第 43 巻, 第 7 号, ダイヤモンド社, 40-52 頁。）

Coplien, James O. and Harrison, Neil B.（2004）, *Organizational patterns of agile software development*, Prentice Hall.（和智右桂訳（2013）『組織パターン: チームの成長によりアジャイルソフトウェア開発の変革を促す』翔泳社。）

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（2004）『ソフトウエア企業の競争戦略』ダイヤモンド社。）

Glass, Robert L.（2006）, *Software creativity 2.0*, Developer.* Books.（高嶋優子・徳弘太郎・森田創訳（2009）『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。）

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

James, Michael. and Walter, Luke.（2010）, "Scrum Reference Card", (http://scrumreferencecard.com/ScrumReferenceCard_v1_3_1-jp.pdf), 2020.1.21.

Rigby, Darrell K., Sutherland, Jeff., and Takeuchi, Hirotaka.（2016）,"Embracing Agile", *Harvard Business Review*, MAY 2016, Harvard Business School Publishing, pp.40-50.（倉田幸信訳（2016）「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,

第 41 巻, 第 9 号, ダイヤモンド社, 92-106 頁。）

Royce, Winston W.（1970）,“Managing the Development of Large Software Systems”, Proceedings of IEEE WESCON, pp.328–338.

Takeuchi, Hirotaka. and Nonaka, Ikujiro.（1986）,“The New New Product Development Game”, *Harvard Business Review*, Vol.64, No.1, Harvard Business Publishing, pp.137-146.

West, Dave. and Grant, Tom.（2010）,“Agile Development: Mainstream Adoption Has Changed Agility For Application Development & Program Management Professional”, January 20, 2010, FORRESTER REPORT.