アジャイル開発の弱点をどう補うか

橋本　哲也　：株式会社クロスフィールド

1. はじめに

世界に比べ日本におけるアジャイル開発の普及率はやや低いものの、PMI 日本支部の 2024 年度調査によれば、組織的にアジャイル開発を導入済みと回答した割合は 44%に達しており、2019 年の 31%と比べれば近年着実に拡大していると言える。ただしその多くは全面的な適用でなく何らかの形でアジャイルを導入しているとされており、特定のプロジェクトや一部工程に限定した部分導入にとどまっているのが実態のようだ。

業種別に導入状況を見てみると「社会インフラ（43.5%）」や「金融・保険（41.3%）」では高い導入率を示しているが、「小売・外食（13.8%）」や「生活関連型・その他製造（18.6%）」では限定的である。これは社会インフラ、金融・保険業界が「顧客向けデジタルサービスの拡充」や「事業変革のための DX」へ積極的に取り組んでいるためであり、これらの取り組みにはスピードと柔軟性が求められ、さらにはビジネスモデルそのものを変革する必要性もあるため、アジャイル導入を後押ししていると考えられる。

ここ数年で筆者が経験したプロジェクトにおいては、大規模システム導入プロジェクトでは依然としてウォーターフォール開発を採用しているクライアントが多かったが、中・小規模のシステム導入プロジェクトではアジャイル開発を採用しているクライアントも珍しくなくなってきた。

本レポートでは、SIer 出身のコンサルタントである筆者が、実際にアジャイル開発を採用したプロジェクトでアジャイルの弱点をどのように解消したかをプロジェクトの実体験をもとに紹介する。

2. アジャイル開発の概要と弱点

もはや説明不要だと思うが、アジャイル開発の概要について簡単に説明する。アジャイル“Agile”とは「素早い」「機敏な」という意味であり、システムやソフトウェア開発において、顧客に価値のある製品を早く、継続的に提供するためのアプローチの総称である。

従来型のウォーターフォール開発と比較すると以下のような違いがある。

項目	ウォーターフォール開発	アジャイル開発
進め方	直線的・段階的	反復的・漸進的
計画	最初に詳細まで決定	流動的で段階的に詳細化
変更対応	困難	柔軟（変更を前提とする）
リリース	プロジェクトの最後	小さく早く反復リリース
重視するもの	計画、ドキュメント	利用者の反応、動くソフト
向いているプロジェクト	大規模、要件が明確で変更が少ない	小規模、要件が変化しやすい、不確実性が高い

このような特徴を並べると全体的にアジャイル開発の方がよく見えがちだが、もちろん弱点もある。よく知られている主な弱点としては以下が挙げられる。

大規模プロジェクトには不向き…関係者が増えるほどチーム間の連携や意思決定の同意に掛かるコストが激増するため、大規模プロジェクトでは管理不能に陥り易い

全体像が見えない…最初に計画を固定しないため、最終的なゴール、総コストや完了時期の見通しを立てることが難しい

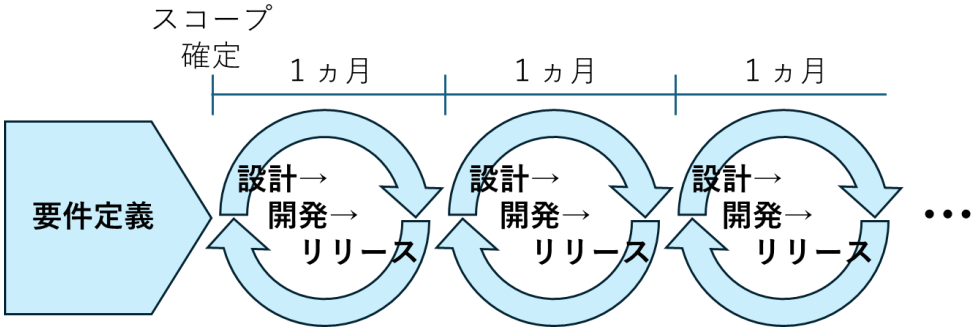
ドキュメント不足…動くソフトウェアの早期リリースを重視するため、仕様書や設計書が不十分になりがちで、保守・引継・システムリプレイスの際に問題となる
技術的負債の蓄積…早期リリースを優先するあまり、保守性や拡張性を犠牲にした暫定対応を多くすることで、将来的に全面改修や開発効率低下の恐れがある
顧客側の重い負担…顧客（発注者）もチームの一員として積極的・継続的に参加しないと意思決定の遅延や方向性のブレに繋がりプロジェクトが破綻するリスクがある
経験者の人材不足…チームメンバー全員が自ら考え動く自律的で高い技術スキルが必要であるが、アジャイル開発の経験者は少なく経験不足のチームではプロジェクトに混乱が生じるリスクがある

3. 弱点への対策・対応

前章で挙げた弱点の中で筆者が直面したものは「全体像が見えない」「ドキュメント不足」、「技術的負債の蓄積」であったため、本章ではその３つの弱点に絞って具体的な対策や対応について紹介する。
まず前提として筆者が経験したアジャイル開発プロジェクトは、全て中・小規模プロジェクトであったため、「大規模プロジェクトには不向き」の弱点は該当しない状態であったことを申し添えておきたい。

全体像が見えない

この弱点は、アジャイル開発では計画を固定せず、初期段階では重要項目のみを精査して細かいスケジュールを組まないため、リリース計画が流動的となってしまうことや変更（要件・機能など）を前提とした進め方のため、スコープがどんどん広がってしまうことが原因であると考えられる。
その結果「最終的にいつ全てが終わるのか」「コストはいくら掛かるのか」などプロジェクトの全体像が開始時点で把握できなくなってしまう。
とは言え、プロジェクトは有期であり、プロジェクト開始前には予算取りも必要だ。筆者も一番頭を悩ませたのがプロジェクトスケジュール作成であった。
そこで筆者のプロジェクトではウォーターフォール開発とアジャイル開発を組み合わせた「ハイブリッド型開発」を採用した。具体的な進め方としては、上流工程の要件定義まではウォーターフォール開発を採用し、設計以降はアジャイル開発を採用した。
要件定義をウォーターフォールで実施することで、全体の要件ボリューム、優先度が把握できるため、「最終的にいつ全てが終わるのか」「コスト（工数）はいくら掛かるのか」がアジャイル開発に比べ初期段階で明確に提示することができた。
また一般的なアジャイル開発サイクルは 1～2 週間と言われているが、筆者のプロジェクトでは 1 ヵ月と少し長めにすることでリリースする機能単位を大きくしたため、いつ何をリリースするかのリリース計画も立てやすくなった。



ドキュメント不足

ウォーターフォール開発では膨大な仕様書や設計書を多大な時間を費やして作成するが、アジャイルではこれを無駄な時間と捉え、動くソフトウェアそのものをドキュメントの代替として扱うとされている。この価値観・考え方がこの弱点の原因であると考えられる。

よく勘違いされがちだが、アジャイル開発は「ドキュメントを作成しなくてもいい」わけではない。アジャイル開発も「要件定義～設計～開発～リリース」という基本的なシステム開発のプロセスは同じで、このプロセスを短期間に何サイクルも繰り返すことがアジャイル開発であり、決して設計フェーズ（設計書作成）を省略するということではない。

ドキュメントを作成しないと保守時のシステム改修に想定以上にコストが掛かったり、新規メンバーへの引継が困難になり属人化の温床になったり、次期システムへリプレイスする際の仕様ブラックボックス化に繋がるため、アジャイル開発であっても可能であれば高品質なドキュメントを作成しておくことを推奨する。

筆者のプロジェクトでは、まずは開発に必要な最低限のドキュメントのみを作成し早期リリースを優先した。何を必要最低限のドキュメントとするかは、プロジェクト計画時にドキュメントの種類・記載粒度を開発チームと相談し、その結果、開発サイクル内で作成できる範囲のボリュームに制限することを決定した。

また開発チーム内に後追いで設計書を作成する担当者を配置し、リリース後一定期間を置いてから、その担当者が設計書を肉付けしていく運用とした。リリース後一定期間を置いたのは、リリース直後の要件変更によるドキュメント修正コストを抑えるためであり、機能が安定してきた時期に設計書を肉付けすることで、最小限のコストで高品質のドキュメントを残すことが実現できた。

技術的負債の蓄積

アジャイル開発では、動くソフトウェアの早期リリースを重視するため、開発サイクル内での成果を急ぐあまり、全体を意識したシステム設計や品質管理が後回しにされる傾向がある。また変更を前提としているため、柔軟に変更を受け入れるが、ここでも早期リリースを優先し場当たり的な暫定対応をしてしまいがちである。これらがこの弱点の原因だと考える。

技術的負債を放置しておくとシステム内が継ぎはぎ状態となり保守性や開発効率低下を引き起こし、障害や要望による改修が発生した際、想定以上に開発工数が掛かってしまう。余計な開発工数が多少掛かるだけならまだマシで、システムの全体最適化がされていないため影響範囲がどんどん拡大し、最終的に本来の改修箇所だけでなくシステム全面改修に繋がる恐れも秘めている。

筆者のプロジェクトでも、早期リリースを優先したため、まずは全体最適を目指すのではなく機能単体で最適なコーディングをする方針としていた。さらにマルチベンダーであったため、同じような機能であってもベンダー間でコーディングレベルに差が発生していた。

ただマルチベンダーはプロジェクト計画時点で分かっていたことなので、部分最適・コーディングレベル差が発生するリスクについては想定済みであり、このリスクの対策として、ある程度の機能がリリースできたら、定期的にリファクタリングする機会を設けることにしていた。なおリファクタリングはバックログ管理対象にして、しっかりと工数を掛けて対応することで、努力義務でなく必要な改修という位置づけで技術的負債の解消を行った。

また、「ドキュメント不足」によって開発チーム内で要件・設計背景の共有ができず、開発の一貫性が崩れシステム構造が複雑化してしまうことも「技術的負債の蓄積」の一因である。そのため「後追いで設計書を作成」する際には、プログラムか

ら設計書を作成する、所謂リバースエンジニアリングではなく、リファクタリングを想定した設計書を作成する方針とした。これにより後続開発のリファクタリングを効率的に行うことができ、「後追いで設計書を作成」の対応で「ドキュメント不足」「技術的負債の蓄積」という２つの弱点の解消に繋げることができた。

筆者が直面したアジャイル開発の弱点に対する対応は以上であるが、今回紹介できなかった「顧客側の重い負荷」「経験者の人材不足」についても教科書的な対策・対応となるが紹介する。

顧客側の重い負荷

アジャイル開発では顧客（発注者）がプロダクトオーナーとして頻繁にレビュー参加や優先順位付け、要件整理などに関与する必要があるため、顧客側の業務負荷が増大してしまう。対策として顧客側の知見不足や時間的制約を補う「プロダクトオーナー支援※1」という役割を設置することが有効である。

※1 プロダクトオーナー（システムの価値・方向性・優先順位などに最終責任を持つ意思決定者）を支援する役割で、外部有識者や支援事業者（コンサルなど）を起用するケースが多い

経験者の人材不足

経験不足のメンバーが短い開発サイクルでアジャイル開発を行おうとすると、サイクルを回すことが目的となりプロジェクト本来の目的を忘れプロジェクトに混乱をきたすケースもある。

この弱点に関する即効性のある対策は「有識者・経験者の採用」であるが、時間をかけて対策・対応が可能であれば「社内教育・研修の強化」「小規模プロジェクトで試用」をすることで社内の人材を育てていくことも長期的目線では有効である。

余談だが筆者のプロジェクトでもアジャイル経験者は少なかったが、プロジェクトに混乱をきたすことはなかった。これは開発サイクルを１ヵ月と少し長めにすることで時間的に余裕が生まれたため、各開発サイクルを焦らず回すことができたのではないかと結果論だが感じている。

4. おわりに

アジャイル開発の弱点を補う鍵は、「柔軟性」を活かしつつ、ウォーターフォール的な「計画と規律」を戦略的に取り入れることだと考える。それには今回紹介した“ハイブリッド型開発”、“開発サイクルの工夫”、“後追い設計書作成”、“リファクタリング”の採用や、他にもクオリティゲート（工程移行判定）による品質管理の導入などあるが、これらはあくまで一例であり組織やプロジェクトの特性に応じた対策を講じることをお勧めする。そのためには継続的に対策を検討し、常に改善活動を行っていくことが重要である。

アジャイル開発は、昨今の変化の激しいビジネス環境において、顧客ニーズに素早く対応し価値を最大化するための強力な手法で、システム開発だけでなく、その他のプロジェクトでも利用できる推進手段（アプローチ方法）でもあるので、本レポートで紹介した対策・対応が今後読者が取り組むアジャイル（開発）プロジェクトの手助けになれば幸いである。

【参考文献】

2024 年度アジャイルプロジェクトマネジメント意識調査報告書（PMI 日本支部アジャイル研究会）