≪コラム 2 ≫

IBM 流アジャイル開発

― IBM 社内 IT 部門のアジャイル開発の取り組み ―

日本アイ・ビー・エム株式会社
グローバル・ビジネス・サービス事業
アプリケーションマネージメントサービス
アドバイザリー・プロジェクトマネージャ
柿田 文和 Fumikazu Kakita

【プロフィール】
1997 年日本 IBM 入社。IBM の社内情報システム部門である IGA AS にてアプリケーションの開発・保守作業に従事。2008 年より同組織内にてアジャイルの推進を担当。

■ アジャイル開発への期待

市場の競争が激しくなる中で、経営戦略上 IT へのニーズは多様化するとともに、スピーディーなシステム開発が求められるようになってきています。従来は、数年がかりの大型プロジェクトが多かったのですが、今では低コスト、短納期、高品質といった、3 つの事象を達成することが求められています。開発スピードをそれほど重視していないウォーターフォール型の開発ではこうしたニーズを満たすことは難しいことから、よりスピードを重視しつつ品質も維持するアジャイル開発の手法が近年注目を浴びてきています。

IBM の社内システム開発でも、2008 年からアジャイル開発手法を導入し、一定規模以上のプロジェクトを中心に実践してきました。新しい開発手法で不慣れなことから、当初は戸惑うこともありましたが、今ではプロジェクト・マネジャーが積極的にアジャイル開発を適用するほど、一般的に使われるようになってきています。

アジャイル開発の適用を予定している、あるいはこれから適用を検討しようとしているなど企業によってさまざまだとは思いますが、IBM 社内 IT 部門で実践しているアジャイル開発をご紹介し、机上の空論ではなく、実際にアジャイル開発を組織として適用して初めて分かった事例などをご紹介します。

■ アジャイルとウォーターフォールの比較

アジャイル開発は、ウォーターフォール開発と考え方に大きな違いがあります。テクニック的な内容もそうですが、本コラムで述べている違いを最初に整理しておきます（表 1）。

■ プラクティスの選択の柔軟性

アジャイル開発手法には、さまざまな方法があります。有名なところでは、XP※1（eXtreme Programming）や SCRUM※2 といった手法がありますが、IBM のアジャイルでは、SCRUM を中心とした反復開発のメソッドを確保しながら、幾つかのプラクティス（実践原則）を組み合わせてアジャイル開発を行っています。プラクティスとは、テスト駆動開発やペア・プログラミング、ストーリー・カード、継続的インテグレーションなどを指します。初めてのアジャイル開発では、どのプラクティスを適用すべきかに悩むと思いますが、その答えはありません。プラクティスの選択は、適用するプロジェクトやアプリケーション特性に応じて、柔軟に考えるべきでしょう。アジャイル開発の本質は、開発生産性を高めて開発スピードを上げることです。プラクティスを適用すれば必ず開発生産性が上がるというものではなく、プロジェクト特性を見極めた上で効果があると思われるプラクティスを適用することが肝要です。IBM のアジャイルでは、プロジェクト・マネジャーにプラクティスの選択の余地を与え、チームの生産性を最大限高める責務を課しています。

表1. アジャイル開発とウォーターフォール開発の比較

	アジャイル	ウォーターフォール
思想	人重視	プロセス重視
開発スコープ	優先順位により柔軟に決定	契約によりスコープを決定
問題の識別	早期に問題識別が可能	プロジェクト管理による
要員の育成	早期に育成が可能	開発段階で集中的に育成
基盤構築	基盤を構築するタイミングが難しい	設計段階から作るのが一般的
オフショア開発の適用	困難にする制約があるため十分検討が必要	開発局面は適用しやすい

従来のプロセス重視の組織には、アジャイル開発のような柔軟性のあるプロセスを新たに適用することをリスクと考えてしまいがちです。しかし、アジャイル・マニフェスト [1] でもいわれているように、アジャイル開発はプロセスを重視するものではなく、人と人との交流を重視すべきです。アジャイル・マニフェストをきちんと理解し、プロセス偏重になり過ぎないように、アジャイル開発を組織に適用していく必要があると思います。

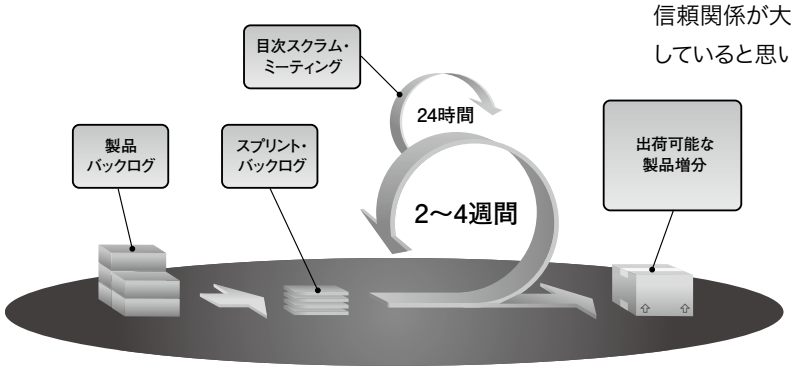
※1　XP：1996年にケント・ベックが開発したメジャーなアジャイル開発手法の1つ。
※2　SCRUM：2〜4週間の開発を繰り返し行い、成果物を徐々に積み上げていくアジャイル開発手法。毎日短時間の会議を行う日次スクラムが有名。

■ お客様の投資対効果を最大化する

アジャイル開発のプラクティスは、開発チームの生産性を高めることに大きく寄与します。しかし、いくら開発チームが高いパフォーマンスを発揮できたとしても、価値のないアプリケーションを作ったのでは、結果としてビジネスとしてのメリットがありません。単にプラクティスを取り入れただけでは、満足する結果を得られないということです。IBM のアジャイルでは、常に「お客様が受けられる価値」にフォーカスし、アプリケーションの開発を進めていきます。お客様が受けられる価値とは、投資対効果（ROI）を最大化することを意味します。この投資対効果を最大化するための方法として、中核プロセスに SCRUM を採用しています。

SCRUM の考え方そのものは、製品バックログと呼ばれる開発したい案件のリストから、優先度の高い順に 1 回の反復で開発可能なものを抜き出し、少しずつ製品バックログを減らしながら開発していくというものです（図 1）。IBM のアジャイルでは、製品バックログの優先順位付けの方法がより詳細に定義されており、投資対効果を最大化することを求めています。一例ですが、一人のユーザーからの強い要望と、数万人が恩恵を受ける要望とでは、より多くの人が恩恵を受けられる開発を優先しなければなりません。また、社内部門からの要望よりも、売上が拡大する開発を優先したりします。常に開発の優先順位を決めるのは、ビジネスにとっての投資対効果が高い順番です。

製品バックログの優先順位を決めるのは、プロダクト・オーナー（お客様）の役割ですが、初めてのアジャイル開発では困難を伴う作業です。ウォーターフォール型の開発に慣れてしまっている場合、要望の一覧を挙げることには慣れていますが、投資対効果が高いという視点で順位を決めることに慣れていないからです。そこで、投資対効果を最大化していくためには、スクラム・マスターによるアドバイスが欠かせません。スクラム・マスターは、お客様と開発チームの第三者の立場で、常に投資対効果を最大化できるためのアドバイスをしながら、最適な要件を選択肢して開発を進めていきます。



出典：Mountain Goat Software, An Overview of Scrum for Agile Software Development [2]

注: スプリントと反復は互換性があり同じ意味

図.1 SCRUM プロセス

アジャイル開発を行う場合、「投資対効果の高い要件の選択」と「生産性の高い開発チーム」という組み合わせで、最高のパフォーマンスを発揮することができます。

■ 新規アプリケーションでのアジャイル適用

まったく基盤のない新規アプリケーションの構築では、アジャイル開発の適用は難しいと考えられています。IBM の社内でも、有識者がいないとアジャイル開発は難しいという認識が一般的でした。しかし、新規開発でアジャイル開発を採用するのは難しいという考えは、大きな誤解だと思います。あるとき、短納期で新規アプリケーションを構築する必要性が生じ、開発方法論を検討した結果、アジャイル開発を適用することになりました。開発メンバーは、誰一人として業務知識を持っていない状況で、大枠の要件しか決まっていない、非常にリスクの高いプロジェクトでしたが、短納期という要求を満たすためには、要件を決めながら開発を進めていくアジャイル開発以外は選択肢として考えられませんでした。

アジャイル開発で実施した結果、新規アプリケーションにもかかわらず計画通りにサービスインすることができました。そして、開発にかかわったメンバーは、アジャイルだからこそ計画を達成できたと口々に言っています。ウォーターフォールのように、詳細な要件を決めてから進めていたのでは、まったくスピード感が合いませんでした。

どんなプロジェクトでも、多少のリスクは付きものです。動く成果物を主体に作って行くアジャイル開発だからこそ早期にリスクを識別でき、問題に対しても素早く対処することができます。新規アプリケーションのようなリスクのあるプロジェクトこそ、素早く行動し問題を早めに発見・解決することで、結果として短納期を達成することができると考えています。しかし、ウォーターフォール開発に慣れている環境では、あいまいな要件のまま開発を進めることを嫌う傾向があります。お客様や開発ベンダーが双方にリスクを認識し、お互い最高の結果を得られることを信じてプロジェクトを進めていく必要があります。アジャイル開発では、お客様と開発チームの信頼関係なくして成功はあり得ません。契約による縛りよりも、信頼関係が大事といわれるアジャイル開発の大きな特徴を表していると思います。

■ アプリケーション基盤は初期の反復に

教科書通りのアジャイル開発では、要件定義を進めながら開発・テストを行い、成果物を見ながら反復して開発を行っていきます。しかし、新規アプリケーションの場合は、データベース周り、ロギング処理、メッセージングなど、アプリケーションの基盤が構築されておらず、すぐにビジネス・ロジックの実装ができません。アジャイル開発に

成熟したチームであれば、基盤を構築しながらビジネス・ロジックの実装ができるのかもしれませんが、失敗すると開発者ごとに実装方法が異なってしまったり、複数人で同じロジックを実装してしまったり、かえって開発生産性を落としかねない事態になってしまいます。

新規のアプリケーション構築では、まずは反復期間の開始前、もしくは初期の反復期間でアプリケーション基盤を構築することをお勧めしています。初期の反復では、まだすべての要件が確認されているわけではありませんので、完全な基盤を構築するのは不可能です。しかし、今後ビジネス・ロジックを実装していくための基盤を作る程度のことはできます。最低限の基盤を構築したら、後は必要に応じてリファクタリングを行い、基盤の更新を行っていけばよいのです。ウォーターフォールでも、最初にしっかりと基盤を作ると、その後一切の更新が入らないということは少ないと思います。アジャイル開発の場合は、ビジネス・ロジックの実装を行い、動くアプリケーションを作るのが肝要ですから、基盤構築に時間を掛け過ぎず、適度なレベルで構築しておくのがスピード感と安定性を両立させる鍵といえます。

■ 開発要員は実践を通じて育つ

ウォーターフォール開発では、一般的には開発やテスト局面で大量の開発者を投入して、アプリケーションを作り上げていきます（図 2）。しかし、アジャイル開発では常に一定の要員をアサインしてプロジェクトを進めていきます。アジャイル開発では、アプリケーションや業務知識を有した開発要員がいないと難しいというイメージがありますが、必ずしも有識者がいなければいけないという訳ではありません。有識者がいた方がよいのは、ウォーターフォール型でも同じことです。

アジャイル開発の場合は、プロジェクトの初期の反復から開発要員をアサインします（図 3）。たとえ最初は業務知識がない開発要員でも、反復を繰り返すごとに知識はどんどん付いてきます。プロジェクト終了時には、立派な担当者として知識を付けていけます。また、同じ開発要員が一定期間アサインされるので、プロジェクト内に知識が蓄積されやすい環境となります。ウォーターフォール型の開発では、開発・テスト局面に山があるので、短期間のアサインで終わってしまう開発要員が多くなりますが、アジャイルの場合は長期間アサインされる開発要員が多いのもメリットです。最初から有識者がいる前提でアジャイル開発を考えるのではなく、上流工程や開発の実践を通じてプロジェクト内に早期に知識を有した人を蓄積していくと考えることが、人を中心に考えるアジャイル開発では大切なことです。

また、ソフトウェア開発は作ったら終わりではなく、その後も維持していかなければなりません。アジャイル開発では、単なるプログラマーではなく、上流工程にかかわった人を育成できるため、その後の保守活動にまでわたって効果を得ることができます。

■ 生産性と品質を高める振り返り

アジャイル開発では、反復開発を繰り返していきますが、初期の反復ほど混乱が生じやすくなります。ウォーターフォール型でも、プロジェクト・チーム結成直後や、開発局面の初期はさまざまな混乱が生じますが、アジャイル開発の場合は一定期間で成果物が求められるため、より混乱の度合いが可視化されやすくなります。一般には、反復を繰り返すほど、開発要員が成熟していくため、開発生産性が高まっていく傾向が見られます。ここで大事なのが、各反復の最後に行う“振り返り”です。振り返りでは、反復期間中に起こった出来事から、継続すること、新しく始めること、止めることをチーム全員で意見を出し合います。ウォーターフォール型のプロジェ

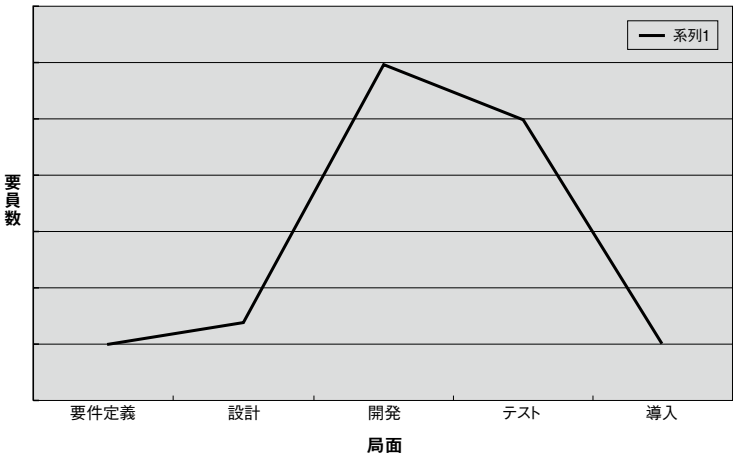


図2. ウォーターフォール型の要員計画

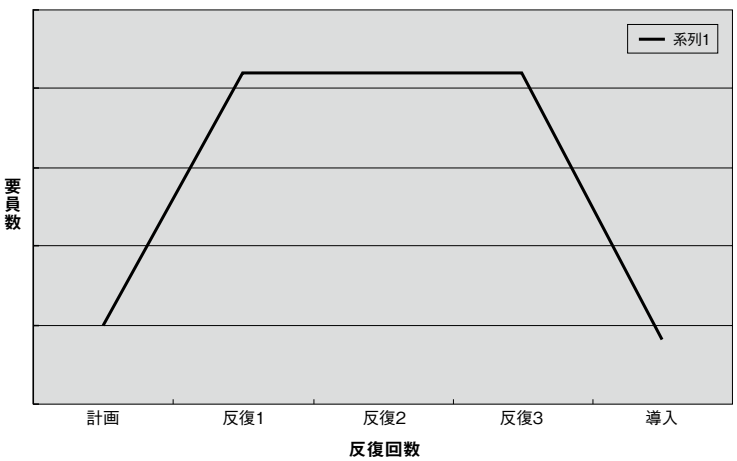


図3. アジャイルの要員計画

クトでは、プロジェクトの最後に反省会を行うことが多いですが、残念ながらプロジェクト・チームはすぐに解散してしまうため、反省会の結果を生かす機会がありません。しかし、反復開発では、振り返りの結果を次の反復ですぐに生かすことができます。例えば、反復期間中に参照されなかった文書の更新を止めたり、テスト効率向上のために新しくテスト駆動開発のプラクティスを取り入れるなど、チームの生産性を高める工夫が出されます。こうした振り返りを行うことで、最高のパフォーマンスを出せるチームが出来上がっていきます。最初の反復から、最高のパフォーマンスを発揮することはできません。開発チームも柔軟に変化しながら、最高のチームになっていきます。

■ オフショアを活用したアジャイル開発

アジャイル開発では、スピード感のある開発を目的としていますが、一方で低コスト開発という観点も欠かせません。IBM のアジャイルでは、オフショアを活用してアジャイル開発を行っています。XP に On-Site Customer というプラクティスがあるように、コミュニケーションを重視するアジャイル開発では、お客様に近い環境で開発をすることが鉄則です。しかし、オフショアを活用した場合は、物理的な距離が大きな制約となります。アジャイル開発の生産性を高めるためには、中間成果物（例えば内部設計書などのドキュメント類）を極力減らしてコードを書く時間を増やさなくてはなりません。中間成果物に代わる部分をコミュニケーションでカバーしていくのがアジャイル開発のやり方です。オフショアを活用した場合は、どのようなプロジェクト体制を組み、効率よくコミュニケーションしていくかが、プロジェクト成功の鍵を握っています。

反復初期のころは、オフショア要員も十分な知識を有していないことがほとんどです。日本側の要求やシステム仕様を、中間成果物を使わずにオフショア側に正しく伝えていくための工夫が必要になります。最も効率がよいのは、オフショア要員の中核となる人を短期間でも日本に呼び、日本側のチームとして参画してもらいながら、オフショア側とのコミュニケーション・ブリッジの役割を果たしてもらうことです。電話会議では、言葉の壁に加えて仕様の理解という二重の難しさがあります。コミュニケーション・ブリッジの人が、的確にネーティブな言葉で仕様を伝えることで、オフショア側の要員が効率的に動ける体制を組むことがその困難を乗り越えることにつながります。

■ Rational Team Concert ™の活用

コミュニケーション・ブリッジと加えて、オフショア要員との円滑なコミュニケーションに、Rational Team Concert（以下、RTC）を活用しています。RTC は、アジャイル開発を行うために最適な構成管理・変更管理・作業管理などを行うプラットフォームであり、開発チームは RTC 上ですべての作業を完結することができます。アジャイル開発に限らず、オフショアを活用する場合は、離れた場所で作業している人を管理しなければなりません。特にアジャイル開発のように、コミュニケーションや開発を重視する場合は、オフショア要員の作業進ちょくをタイムリーに確認できる必要があります。定期的な進ちょく会議だけでの確認では、開発のスピード感を大きく低下させてしまいます。誰がどのタスクを作業しているのか、どんなコードを修正中なのか、全体の進ちょくに遅れはないのかなど、開発工程の管理に必要な要素を、オフショアの開発者は RTC のプラットフォーム上で作業するだけで、タイムリーに日本側から状況を確認することができます。

先にも触れましたが、IBM のアジャイルでは、オフショアを戦略的に活用しています。そして中国やインドのような新興国では、20 代の若い世代の人たちがほとんどです。これらの若い世代はインターネット上の仮想ゲームなどに代表されるように、チャットやタイムリーな情報を駆使しながらお互いに協調し、時には競い合う、バーチャルな世界でのコミュニケーションに慣れています。オンサイト中心での開発に慣れ親しんでいる日本から見ると、オフショア開発には不安があるかもしれませんが、オフショア要員の方がバーチャル環境を使いこなすのは上です。RTC は、アジャイルの思想を取り入れただけではなく、オフショア要員の競争力を引き出してくれるプラットフォームなのです。

■ おわりに

IBM のアジャイルは、単にアジャイルのプラクティスを活用するだけではありません。お客様の価値を最大化することにフォーカスし、人間工学に基づきチーム・パフォーマンスを最大化するための考え方が盛り込まれています。かつ、オフショアを活用したアジャイル開発で、低コストでのアプリケーション開発も目指しているのが大きな特徴です。

アジャイル開発そのものは、組織やプロジェクトによって、実施形態はさまざまに変化するため、やり方について絶対的な答えがあるわけではありません。本書に記載した内容も、IBM が考えるアジャイルの定義の 1 つに過ぎません。アジャイルの精神に基づき、計画中心の文化から、変化を柔軟に受け入れられる人が中心となって、変化に強いチームを作っていくことが必要になります。一定のルールの下にまずはやってみて、それから考えるという発想がアジャイル開発には求められるのではないでしょうか。

［参考文献］

［1］ Agile manifest, http://agilemanifesto.org/
［2］ An Overview of Scrum for Agile Software Development, Mountain Goat Software, http://www.mountaingoatsoftware.com/scrum/overview