アジャイルソフトウェアプロセスを使って オフショア開発

マーチンファウラー著、金田忠士訳

http://martinfowler.com/articles/agileOffshore.html

2003 年 9 月

概 要

これまでの 2 年間、ThoughtWorks 社は、北米やヨーロッパでの我々のソフトウェア開発プロジェクトを支援するために、インドはバンガロールに研究所を稼動させてきた。オフショア開発への従来のやり方は計画ドリブンな方法論に基づいているが、我々はアジャイル推進派だ。本稿では、我々がオフショアアジャイル開発の中での経験と学んだことについて論じよう。これまでのところ利点についてはまだまだ議論の余地はあるが、オフショア開発においてアジャイル開発が有効であることが分かった。

目 次

1

いろんなアジャイルソフトウェア方法論の根本的教義のうちの 1 つは、ソフトウェア開発に関わる様々な人々の間で行われるコミュニケーションの重要性である。さらに、アジャイル手法では、フェイスツーフェイスでのコミュニケーションを通じてコミュニケーションを改善することをとても奨励する。アジャイル宣言でも「開発チームに対して、あるいは開発チーム内部で情報を伝えるもっとも効率的で効果的な方法は面と向かって話をすることである」と述べている。エクストリームプログラミングでは、チームが緊密になっていっしょに仕事ができるようなオープンな 1 箇所に開発拠点を置くというプラクティスでこの点を強調している。コーバーンの本アジャイル手法においては、物理的に近いということの重要性について述べることに多くを費やしている。

近年、ソフトウェア開発業界がとりこになっている他の傾向は、より低い賃金の開発要員を多く調達できるコスト削減に有効な国々でのオフショア開発への移行である。オフショア開発はいくつかの点でアジャイル開発には向いていないと思われていた。まず第一に、オフショア開発は遠くはなれて実施するという定義により、物理的に近くという考え方にすぐさま反する。第二に、ほとんどのオフショア組織は計画ドリブンな手法を好み、構築したい要求仕様の詳細や設計をオフショアに渡している。

従って、基本となる問いは、アジャイル手法がオフショア環境で駆使することができるか?である。もしそうであれば、それは計画ドリブン手法 (私がここで非アジャイルを表すのに用いる用語) を使うのと比較してどうなのだろうか?

私がここに書いている経験は、ThouhtWorks 社によって過去 2,3 年に渡って行ってきた仕事に基づいたものである。我々は 2001 年にインドのバンガロールにオフィスを開き、バンガロアに本拠を置くチームを用いていくつかのプロジェクトをこなしてきた。さらにメルボルンオフィスとのオフショア開発も行った。これらのプロジェクトで、我々はできる限りアジャイルアプローチを用いて係わってきて、その結果我々はアジャイルが顧客の利益を最大化するアプローチであると信じた。このエッセイでは、これまでに得られた教訓のうちのいくつかについて述べることとする。

状況を分かっていただくために、バンガロールオフィスを立ち上げた時のやり方については述べるに値すると思う。我々はこのオフィスがオフショア開発環境の第一拠点として用いられることを期待し、そしてほとんどのアプリケーション開発者を新規に募集した。大多数は大学の新卒者だが、何人かの開発経験者も取り混ぜた。我々が非常に高いレベルの採用基準を維持できた (200 人中 1 人にしかオファーを出さなかった) ことは非常に重要な点であり、このモデルはインドでその後も継続した。その結果、我々は非常に優秀な、様々な経験レベルをも取り混ぜた開発者グループを持っている。また、よ

りざまざまな経験を積んだ US 開発者も呼び寄せ、ソフトウェア開発ややアジャイル／ XP プラクティスの経験の浅い開発者たちのメンターとし、とても楽しんでいる。現在、バンガロールには約 40 人の開発者を抱えている。

メルボルンオフィスはまた違った話がある。ここに、オーストラリアの仕事の第一拠点になることを期待してオフィスを設立した。オフィス立ち上げ当初、我々はオーストラリアの仕事を得ようとしている間、いくつかのオフショア開発を行った。チームは同じように才能があったが、我々は多くの経験を積んだ開発者を雇う傾向にあった。

2 教訓

2.1 インテグレーションの悩みを解消するために分散継続的インテグレーションを利用せよ

私は、複数サイトのチームを束ねて仕事を進める際の問題に関する話をいくつか聞いた。インタフェースの定義に多くの注力を払っていても統合する際の問題は頭をもたげてくる。多くの場合これは、インタフェースの意味を厳密に指定するのが非常に難しいからだ。従って全てのシグネチャを正しく指定しても、実際にどのように実装が行われるかの仮定で躓いてしまう。

実際に一番早く ThoughtWorks 社に根づいた XP プラクティスは「継続的インテグレーション」だった。また、我々は継続的インテグレーションに慣れてきたので自分たちのオフショア開発に用いようと決心していた。はじめから我々は全員を 1 つのソースコードを基にさせ、コードのビルドとテストの実行を両方がはしる CruiseControl を使った。この方法は全員が例えどこにいても主流の近傍に維持しつづけることができる。

このやり方がどれぐらい上手くいくかを誰もが楽しみにしていた。他のグループを悩ます問題が自分たちに降りかからずに結合できるということが一番の利益なのだ。継続的なインテグレーションとテストというプロセスは結合の際に出てくる多くの問題を即座にあぶり出し、その結果問題を見つけるのが困難にならないうちに修正することが可能になる。

クルーズコントロールのウェッブページでは、全てのサイトから他のサイトで何が起っているかを確かめることが可能である。朝まず最初にクルーズコントロールのウェッブページを見て、他のサイトでどういう変更が行われたかを確認する。このことで世界中の他のサイトで起っていることにキャッチアップする簡単な方法が提供されるのである。

こうすることでいいビルド規律を要求することになり、そのため開発者たちはビルドを壊すことなく、また壊すことになる場合はすぐさまに修正するように一生懸命努める。開発の主流に変更をコミットする場合、クルーズコントロールからその変更した結果のビルドが成功したというメールを受信す

るまでは帰宅してはならない、というのは一般に容認されたプラクティスである。リモートオフィスで同じビルドが止まってしまうので深夜にビルドが失敗することはたいへん重大なのである。

世界中で継続的インテグレーションは人気を博してはいるが、我々はいくつかの問題にぶち当たった。通信路は我々が気に入るほど太くなくまた信頼性にも欠ける。だから、ソースコード管理の作業によってリモートサイトが不便を強いられてしまう。一般に我々は、ビルドサーバを一番開発者の数が多いサイトと同じところで運営するが、リモートサイトでは、開発の主流から最新版を取り寄せるためにとても時間がかかることを知ることが可能だ。通信経路が遠ければ遠いほど、通信不調から通信路のダウンまで何かが起る可能性が高くなる。リポジトリへ 24 時間アクセスできるようにすると、バックアップのためのシステムダウンに悩ませることになる。これらの問題は全て、ソースコードリポジトリをクラスタ化することによって緩和することができるが、まだそのような実験は行っていない。

(興味深いことに、人々はこれらのコミュニケーションの問題は、インドのような遠くのリモートサイトだからこその問題であると考える。しかし、我々は西側のインフラでも同じような問題がしばしば生じるということが分かった。継続的インテグレーションには優れた接続性、しばしば人々使っているものよりもさらに優れた接続性が求められるのである。)

2.2 各拠点からは別の拠点に常駐者を送り込め

前述したように、アジャイル手法では人と人とが向かいあって対話することが重要であると強く主張する。例え全員が同じロケーションにいなくても、何人かを異動させれば得られるものは明らかだ。当初から我々は US チームの誰かが常にインドに滞在してコミュニケーションを促進させる、ということを決定していた。このような駐在員は US に本拠を置く人々をよく知っており、全員がコミュニケーションをとるのを手助けできる性格を持っていることだ。

我々は今ではこれをいくつかのレベルに拡張している。我々は、米国の開発者や米国のアナリストをインドへ送り込み、技術的な、あるいは仕様レベルのコミュニケーションを取ることはとても有用であるということが分かった。さらに、インドから誰かを US チームに招くのもまた有用である。コミュニケーションが改善されることで飛行機代ぐらいはすぐにペイするのだ。

ビジネスの観点で見たオフショアチームの大使の利点の 1 つはオフショアチームにビジネス状況を伝えやすくする、ということだ。要求仕様のリストからたった 1 つ抜け落としてソフトウェアを開発すると、ビジネス上の背景が大量に抜け落ちてしまう。開発者というものは何をするべきかは聞かされるがなぜそれが重要なのかは聞かされない。このために、適切な判断にしば

しば大きな食い違いが生じてしまうのだ。

駐在員の任務の中で重要なものはゴシップを伝えるということだ。プロジェクトには非公式なコミュニケーションが多くなされる。ほとんどは全然重要ではないことだが中には重要なものがあり、またどれが重要なのかは分からないという点が問題なのである。だがら駐在員の任務には、正式な連絡経路では全然重要とは思えないようなちょっとした、多くの連絡をすることが含まれるのである。

我々は通常、駐在員を数ヶ月ごとに交代させる。こうすることで駐在員 (彼らはあまり長期間母国を離れたがらない) を派遣しやすくなる。また、駐在員としてその期間を過ごすことで、リモートチームにのことがよく分かるようになるのである。駐在員を人選する際に、個人の要望や嗜好に注意を払うことはとても重要だ。数ヶ月の間、家から数千マイルも離れたところで過ごしたくない人もいるので、そういう人を駐在員として選んではいけないのだ。

さらに我々はプロジェクトマネージャを時々駐在員として派遣することが重要である、ということが分かった。プロジェクトマネージャの仕事のほとんどは衝突を解決し、問題が深刻になる前に摘み取りやすくすることだ。電話のどちら側でも働いたことがあるという経験はプロジェクトマネージャにとってそういう仕事を効果的に行うためには本当に重要だ。

2.3 文化を変えることを過小評価するな

アジャイル手法を組織に導入する際にもっとも難しいことの 1 つは、このことで引き起こされる文化的な変化である。確かに、我々は、これがアジャイル手法を採用する組織がもつ問題である、ということは分かっている。多くの会社では、上役者が重大な判断を行い、下のものがそれを実行するという仮定のコマンドコントロールモデルで動いている。アジャイル手法を実施するためには、実行者による自律と意思決定を必要とする。

我々はこれが、欧米企業の中で重大な問題になることは分かるが、アジアの文化では目上に従う傾向が強いのでこの問題はアジアの方がより深刻なものとなる。(私が以前に見学したインドの有名な建設会社でのトレーニングコースでは、マネジメントを「コントロールに関する科学」であると定義していた」。) このような環境では、人々はしばしば　質問を投げかけたり、問題点について話をしたり、デッドラインを守れないことについて警告を出したり、目上から認められた指示に対して代案を提案したりといったことは文化的に抵抗があるのだ。

このために、チームを事前の対策に講じさせるのは困難だし、そのために多大な時間がかかることはよくないことだ。例え彼らが見抜いていてもあなたには問題が起っていることなど想像もできないのだ。人々を分散管理のマネジメントスタイル下に置くのは想像以上に時間がかかるのだ。

しかし、いい知らせもある。一度、メンバには意思決定を行う自由と責任があるということが分かるということだ。彼らは本当にそれを楽しんでいるように見える。我々のインドのチームの何人かは、他社で働く彼らの友人たちには、自己裁量が与えられていることがどれだけ信じられないか話してくれた。この自己裁量は大きなモチベーションであり、これによってよりよい生産性、より能力の向上にをもたらせてくれるのである。私にとって我々が発見する最も興味深いことの 1 つは、長きにわたるアジアと西洋両方の文化的影響なのである。

これらの文化的な問題についてさらに話を続けることは問題につながるかもしれない。本稿のドラフトに目を通した何人かの (西洋人の) 産業アナリストたちは、この章は恩着せがましく攻撃的だと言った。バンガロールにいる我々の開発者の一人は、私が穏やかすぎると言った。他には、それが問題であるとコメントしたが、多くの西洋の企業が悪いのかどうか訊いた。しかし、アジアでは、命令と管理を強要する文化的力があるという点にはコンセンサスが得られるように思う。

(これは、ThoughtWorks 社にとって特に際立った問題なのだ。我々は米国内でさえ強い印象を与える反権威主義であると公言している。我々はインドでも同じ文化を保とうと最初から決めていた。私は確かに成功しているようだと言えてとても嬉しく思っている。

2.4 要求仕様を理解するためにテストスクリプトを利用せよ

距離が遠く離れれば離れるほど、要求仕様を伝えるにはより多くの儀式を盛り込まないといけない。我々は単独サイトでの開発で利用する多くの技術に未だにしがみついていながらにもそれらを伝えることができたのだ。

私はますますより多くの成熟した XP チームが、要求仕様を伝える手段として受け入れ試験を用いる、ということを知った。このようなチームでは、イテレーションを始めるまえに試験スクリプトを書いて要求仕様を明確にし、開発チームに確固たるターゲットを伝えやすくした。よく使われるスタイルの 1 つは、　米国にいる顧客のために、フィーチャ(XP のストーリーに相当)を肉付けして (2,3 ページの) 短いストーリを書くことだ。その後、インドにいるアナリストやテスターがこのストーリー向けのテストスクリプトを作る。我々は自動テストを非常に好んでいるが、このスクリプトは自動テストにも手動テストにも用いることができる。スクリプトが作られると、US とインドのアナリストが E メールや IM、定期的な (週に 2,3 回) 電話会議をしてこのテストスクリプトのレビューを行う。

我々は、この作業がインドにいるアナリストおよび米国にいる顧客の双方が本当に要求仕様を理解するのにとても役に立つということを知った。テストを書き上げることによってインドのアナリストは何が必要とされているの

かを理解し、かつ質問が出てくるとともに、US の顧客に質問をするようになる。開発者たちはテストスクリプトをいじるよりもインドのアナリストに尋ねる方が手っ取り早いということに気づく。よって、インドにアナリストやテスターを置くこともまた重要なのである。サーチエンジンもいいけど、人間の方がもっといい仕事することもあるんだ。

2.5　フィードバックを得るために定期的にビルドを実施せよ

人々がアジャイル手法で集めた要求仕様を検討する際、フィードバックループの重要性を見逃してしまうことがしばしばある。時折、用件定義プロセスではアナリストらが要求仕様を提供し、それが開発者たちの手に渡り、実装を行うように思われている。その後の段階で、アナリストが　開発者たちが求められたものを実装しているかどうかを見るために時折チェックを入れる。アジャイルプロセスでは、顧客と開発者の関係を密接なものにすることによって顧客自ら頻繁に開発の進捗状況をモニタできるようにし、誤解をよりすばやく見つけることができるようにする。さらに、部分的に完成しているシステムを使って顧客への教育を行うことができる。というのも、多くの場合、求められているものと本当に必要なものとの間には差異があるものなのだ。しかし、通常はその差異は動くソフトウェアができるまでは明確にはならないのだ。

定期的にモジュールの結合を行うことによって米国にいる顧客が前夜に作ったものを引っ張り出してきて試してみることが可能になる。同じロケーションで開発を行っている場合ほど即座というわけではないが、顧客は自分の要求仕様に対する理解を洗練させることができるのと同様、どんな誤解でもすばやく修正することができるようになるのである。

この仕事を行うためには、ローカルサイトとリモートサイトの環境を適切に同じにするため、環境上の問題を整理することがは重要なことだ。手元のサイトの担当者がビルドを引っ張ってきて、それに問題がみつかり、でもオフショアサイトの担当者には環境上の問題のためにその問題が再現できないという状況ほどいやなものはない。早く環境を整理し、どんな環境上の問題でもそれが起ったらするに解決できる要員を確保できていることを確認することだ。

2.6　定期的に短時間のステータスミーティングを実施せよ

アジャイル手法では定期的に短い進捗会議 (Scrum ではスクラム、XP ではスタンドアップミーティング) をチーム全体で実施する。これを場所が離れたグループも含めることは重要であり、これによって他のチームとの調和が得られるのである。時間帯はしばしば一番大きな問題であり、特に米国とイ

ンドではどんな時間に設定しても都合がわるい。総合的にみると、週に 2 回スタンドアップミーティングを実施するのが十分な調整が可能になり上手くいくように思う。

時間帯の問題では、電話をつなげる時間を選ぶ際には両者でギブアンドテイクをすることが重要だ。我々のクライアントのうちの 1 つは彼らの営業日にしか電話をつなげず、これはインドの人々にとても都合の悪い時間に参加を強いることとなった。一方にだけ重荷を背負わせるのは、共同作業をスムーズに実施するためには全く有効ではない。

2.7 短いイテレーションを繰り返せ

概して、アジャイル手法では他の多くの反復アプローチよりも短いイテレーションを用いる。ThoughtWorks 社では。プロジェクトのほとんど全てで 1〜2 週間単位のイテレーションを用いている。経験豊富なインドの開発者たちの数人は、2,3ヶ月単位のイテレーションを用いるところで働いたことがあるが、より短いイテレーションの方がはるかに優れていると報告している。

我々は、身近のみでのプロジェクトでは 1 週間単位のイテレーションが有効であるように思われたが、オフショアプロジェクトでは 2 週間単位のイテレーションがコミュニケーションのオーバヘッドを最小限にできるということが分かった。

2.8 遠隔サイトをまとめるためにイテレーションプランニング会議を実施せよ

イテレーションを開始するたびに実施しているチーム全体でのプランニング会議が、我々のほとんどのプロジェクトで、次の週の全員の作業項目を調整するのに役に立つことがわかった。私は、我々のプロジェクトのほとんどで、イテレーションプランニング会議 (IPM) を自分たちのプロジェクトの状況に合わせてバリエーションを持たせるようになったことに気づいていた。(こういった類の自己適応は、アジャイルプロセスにおいては重要なことである。)

遠隔地のチーム、特に、変な時間帯の問題を抱えている場合には、一通りの制約を課すことになる。しかしながら変な時間帯に実施されるミーティング時間がつらいにも係わらず、それでも IPM は非常に有用である、ということがわかる。

IPM を実施する前に、米国の顧客からプロジェクト中にテストスクリプトに替わる、予定されているフィーチャ(ストーリ) が送られてくる。この期間は、多くの質問がメールで飛び交う。IPM の直前に、開発チームはそれらのフィーチャを扱いやすい大きさのタスクへの落とし込みを行う。このタスクへのブレイクダウンはフィードバックに備えて米国側と共有する。

この事前準備のために、タスクブレイクダウンで発生したいろんな問題に専念するための通話が短くなる。我々は、通常、最後の 30 分から 2 時間まで電話を受けるのである。

2.9　コードベースを移すときはバグフィクスから始めよ

我々の 2 件のプロジェクトでは、大規模な (数 100K ライン) コードベースを持っていて、さらにバンガロール研究所にそのコードベースの開発のメインを移すということを行った。このどちらのプロジェクトでもインドのチームは、新しい機能を追加する前に、バグ修正のための数回のイテレーションから開始した。

バグフィクスからやらせると、開発者は変更するよりも多くのコードを読むことになるので、本格的な作業を行う前に、インドのチームがそのコードベースに精通することができた。これはうまくいったが、もっと経験豊富なエンジニアには、バグ修正しかさせてもらえないのは名折れだと考えてしまう、という懸念がある。これを問題だと考える人がいるが、認識バグ修正や非常に局所的なフィーチャの変更というのは、大規模で新しいコードベースに慣れるには一番いい方法のうちの 1 つなのだと私は信じている。

2.10　チームは工程ではなく機能で分割せよ

オンショア／オフショア間の境界についての従来の考え方の多くは、エンジニアたちが作業を行う工程に基づくものだ。つまり、分析と設計はオンショアで行い、構築をオフショアで行い、そして受入試験をオンショアで行う。これはウォーターフォールモデルにとてもよくマッチする。

我々は、これに反して、オフショアのチームにできるだけ多くの工程をやらせると問題が改善される、ということが分かった。だから、我々は可能な限り分析や設計を彼らにやらせ、要求仕様がオンショアから出ることによる限界に委ねてしまいたいと思う。開発チームをオンショアとオフショアに分ける場合、作業工程に沿って分けるのではなく、機能面に沿った分割を行う。システムを大まかなモジュールに分割し、オフショアチームにこのモジュールのうちのいくつかを任せてしまうのだ。でも、他のグループと異なり、モジュール間のインタフェース設計や仕様凍結に多大な努力を払うようなことはやらない。継続的なインテグレーションとコード所有権を重視しないことで、モジュール間インタフェースは開発が進むにつれて発展するのだ。

オフショア開発における重要な点は、オフショアチームのアナリスト担当を育てることだ。同じチーム内の開発者がビジネスに対して理解を深めれば深めるほど、開発チームはより効率的に開発を行うことができるようになる。これは、オフショア側のアナリストのビジネス知識を育てることにフォーカ

スしなければならないことを意味している。これには時間がかかるが、同じチーム内にある知識は、オンショア側にあるビジネス知識に匹敵するほどなのだ。

2.11　多くのドキュメントが必要と心得よ

アジャイル手法では、ドキュメントを作る作業の大部分は無駄に終わるという観察結果から、ドキュメント作成を軽視している。しかしながらドキュメント作成はオフショア開発においては、対面のコミュニケーションを減らすことができるのでより重要になる。この作業はもしチーム全員が同じロケーションにいる場合は必要ないものだから、ある意味無駄である。でも、オフショアモデルだと、これはやらないといけない。従ってこれは、オフショア開発を行う上での必要経費なのだ。ドキュメントを書くことは時間的な経費であり、それは、多くの担当者にとってドキュメントから多くのことを理解するのはとても困難なための時間の追加であり、そのドキュメントを使っているときのフラストレーションなのである。

ドキュメントと同様、もっと便利なコラボレーションツール：Wiki、バグトラッキングツールやそれに似たものがさらに必要になる。だいたい、よりややこしくない、つまりチームがやりたい進め方にあわせられるツールの方が有効であるように思う。(それが、Wiki がいい、という理由の一つだ。)

ドキュメントであろうが、何か別のものであろうが、覚えておいて欲しいことは、他の人のテンプレートはあなたの役には立たず、また最初から望ましいスキームが手に入るわけではない、ということだ。確かにドキュメントの形式や、どれだけそれが便利かという話題はとてもたくさんある。でも、あなたのチームにとってどれが一番便利かが分かるにつれて、ドキュメントの構造は進化することを期待しなさい。

> アジャイルプロジェクトで成功するドキュメンテーションには 2 つ鍵がある。まず 1 つは、「ちょうど十分な」ドキュメント量を見つけることだ。これを決定するのは困難で、プロジェクトごとに様々だろう。幸運にも、アジャイル開発の繰り返しという特徴のおかげで、正解を見つけるまで実験は可能だ。アジャイルドキュメンテーションを成功させる 2 つ目の鍵は、ドキュメンテーションを重要視しないこと、あるいはアップデートし続けられるなんて非現実的な希望をもたないことだ。ドキュメントは特定の目的に役立つために作成されなければならないし、その目的に役立った後は、あなた方は誰もがおそらくそのドキュメントを更新しつづけるよりも大事なことがあるはずだ。これは、普通はにわかに信じがたいが、おそらく今度必要になった時に新しくドキュメントを作った方がいい。プロジェクトで、ドキュメントが必要

になるたびに、それを作り始めることの副作用は、ドキュメント作業を効率的にしておく大きなインセンティブなのだ！–[Simons]

3　オフショア開発におけるコストと利益

一般的なエンタープライズソフトウェアの世界と ThoughtWork では、オフショア開発を用いる際のコストと利益に対する考え方にはまだ多くの意見の食い違いがある。それは、ほとんどの人はオフショアはコストを減らすためものであり、オフショアベンダを探すのに、十分に低い人月単価にしか注目しないからだ。しかしながら、人月単価しか見ないのはばかげている。人月単価はコストの単なる要素の 1 つに過ぎず、どんな場合でも投資効率全体に注目しなければならない。ソフトウェア産業では誰もが、開発者による生産性の違いは、給料の格差よりもはるかに大きいことを知っている、いや知るべきだ。そしてオフショアから提示される人月単価による差額などそれほど大きなものではない。オフショアには単価の低さを相殺してしまうぐらいの導入のための追加コストやリスクが存在するのだ。

最も重要なのはコミュニケーションに対する影響だ。オフショアはその (実際に会うことが困難なほどの) 距離と時差のためにコミュニケーションが非常に難しい。このどちらもが要求仕様を伝える際のコミュニケーションミスにより、間違った機能を構築してしまう可能性を高めるのだ。常駐者を使うといったようなテクニックを減らそうとすると、何らかの影響がまだあるだろう。個人間に人間関係が希薄になるので、開発とビジネスが大きく乖離し、開発チームのモチベーションを落としてしまうのだ。

もちろん、主要なコミュニケーションメカニズムとしてドキュメントを用いるような形式ばった組織ではこの点に関してはそれほど問題視しないだろう。本質的に、彼らのコミュニケーションは既に直接会って話す機会が不足しているので多くの損害を受けているはずだ。従ってオフショアを使うことの影響はそれほど顕著ではない。アジャイル手法ではコミュニケーションを改善するために、直接会って話す機会を増やそうとする。我々の経験では、アジャイルアプローチだとオフショアとのコミュニケーションで苦労してもそれが、ドキュメント駆動のアプローチよりはまだマシだということだ。

この問題を解決するために、別の傾向が役に立つかもしれない。企業はますますビジネスプロセス機能をオフショアへ移動させている。もし企業が会計機能をインドへ移せば、会計機能を西洋に置くよりも会計支援ソフトウェアをインドで作るのがより容易になる。この種のビジネス機能の移動がこれからも続けば、インドでの開発がオンショアの代替案になる。

オフショア開発を実施する別の利点は、市場に出すまでの時間を短縮ために開発に 24 時間使えることだ。これで手にする利点は、一日中コードベースに手を入れることができ、求められている機能を早く書き上げることができ

る。私は、オンショアチームがどの機能を追加しないでインドのチームがどんな機能を追加するのか分からないので、この議論は多少眉唾に見えることを白状しなければならないが。

24 時間開発という考え方にある、面白い情報は、テクノロジの発達が減速しているにも係わらず、才能ある開発者を見つけるのはまだ容易ではない、ということだ。従って、オンショアロケーションでは十分な才能ある開発者では見つけられないことがしばしばある。そこで、オフショアチームというのは比較的低いコストよりもそういった才能を調達するという点で価値があるのだ。

これらの全ての相違点において私の観点は明らかだ：私はどっちつかずの態度を取っているのだから。

4　オフショアとアジャイルのこれから

私がこれを書いたように、オフショア開発は非常に流行っているが、まだ早すぎてその本当の強さと落とし穴についてまだ本当には理解できていない。確かに人月単価の違いと同じコスト削減ができると思ってるから実施しているというのはひどい思い違いである。ソフトウェア開発を第三世界へ持っていくのを鉄鋼業がやったのと同じやり方で話す人もいる。他にも、甘い蜜を吸った後、オフショアの産業は干上がってしまうだろうという人もいる。私の水晶玉には、私の前にあるものだけをわずかに歪んで私に見せているのだ。

一つの結論は明らかだ。オンショア開発の方がスキルが高いので成功するだろう、と考えるのはとても間違っている。我々は北米やヨーロッパにひけをとらない才能をもった開発者をインドで見つけることができる、ということがわかったのだ。

オフショア開発の弱点は、文化面やビジネス拠点からの距離によるものだ。アジャイル開発が密接なコミュニケーションとオープンなカルチャーによって一番有効に働くのでオフショアで働くアジャイリストたちは計画ドリブンなアプローチを使う以上の苦痛を感じる。しかし、計画ドリブンな手法ほど苦痛ではないのだ。

我々はオフショア開発の利点、欠点を決して本当には理解できないかもしれない。ソフトウェア開発というのはアウトプットを比較評価するのが不可能な活動なのだ。我々はあるアプローチが別のアプローチよりも優れていることを証明するための具体的数字を手に入れることは不可能だろう。我々が見ているものは、俊敏さとオフショア開発両方の利点における質的なフィードバックを育てることだ。–この質の評価は、この生き残るどちらかあるいは両方が決めるのである。

5　さらに知りたい人のために

我々のところの一番バンガロール研究所にいたプロジェクトマネージャのマットサイモンは彼の経験から記事を何本か書いた。このうちで一番分かりやすいのが、「Internationally Agile」で informit.com に置いてある。(残念ながら、この記事へのリンクを私は見つけられないでいる)

5.1　更新履歴

本記事に関するメジャーアップデートのリスト

- 2003 年 9 月:第 1 稿