エクストリーム・プログラミングの神秘を解く: 「XP の真髄」に立ち戻る 第 1 回

XP の誇大宣伝を詳しく検討する

Roy Miller
Independent consultant

2002年 8月 01日

Java 言語によるオブジェクト指向プログラミングは、きわめて一般的になっています。この方法はまた、ソフトウェア開発をかなりの程度変革しています。それでも、最近の研究によると、ソフトウェア開発プロジェクトの半分は予定より遅れ、3 分の 1 は予算をオーバーしています。問題はテクノロジーではなく、ソフトウェアを開発する方法にあるのです。Java などのオブジェクト指向言語の能力および柔軟性と、いわゆる「アジャイル」手法を組み合わせることにより、この問題の答えが得られそうです。最も人気の高いアジャイル手法は、エクストリーム・プログラミング または XP と呼ばれるものですが、多くの人々は、それがどういうものなのかを本当には知りません。ソフトウェア開発プロジェクトで XP を使用すると、劇的な成功を収める可能性が高くなります。Roy Miller によるこの新規コラムは、好評を博した彼の記事「XP の真髄」に立ち戻ることから始められ、うわさや誇大宣伝をはぎ取って XP への理解を助け、なぜ XP がそれほど重要なのかを説明します。

このシリーズの他の記事を見る

Chris Collins と私が「XP の真髄」を書いてから 1 年になりますが、そのときから、状況がだいぶ変化しています。エクストリーム・プログラミング (XP) はかなり成長していて、自分たちの組織で XP をインプリメントしている人も、これまで以上に増えています。しかし、XP が支持を集め、それによって活況を呈しても、XP がどのようなものであって、どのようなものでない のかという点については、まだ多くの混乱と議論が見られます。驚くほどのことではありませんが、Microsoft 社も、同社の最新オペレーティング・システムに「Windows XP」という名前を付けて、この混乱に一役買っています。

これからの数か月、このコラムを借りて、XP にまつわる誇大宣伝、不実表示、および正真正銘の混乱を詳しく検討して行く予定です。XP に関する皆さんの疑問に答えるための実際的なリソースと、皆さんの組織で XP をインプリメントするためのアイデアを提供できると期待しています。このコラムの最初の数回の記事では、「XP の真髄」からさらに進んで、XP に関する現行のプラクティスや考え方について述べる予定です。私の目標は、XP をこれまで以上に活用するための十分な基礎を提供し、それが皆さんの会社や皆さん自身のキャリアにいかに重要であるかを説明することです。

私は、ほぼ 10 年間、何らかの形でソフトウェア業界で仕事をしてきました。その間、XP ほど私を興奮させるソフトウェア開発手法を見たり、経験したりしたことはありませんでした。これは、プログラムするための理にかなった方法であると確信しています。最初のうちは皆、このやり方が理にかなっているとは感じ られないかもしれません。しかし、本領を発揮する機会を与えてみると、これまでなぜほかの方法でやれてきたのだろうと思うはずです。

もちろん、優れたソフトウェアを開発する方法は XP だけではありませんが、次の 2 つのことは間違いないと確信しています。

- XP がプラクティスに組み入れる原則は正しい 原則である。
- 私は、XP ほど、あらゆる事柄を一緒にまとめて行うことのできるソフトウェア開発手法を、これまで経験したことは決してない。

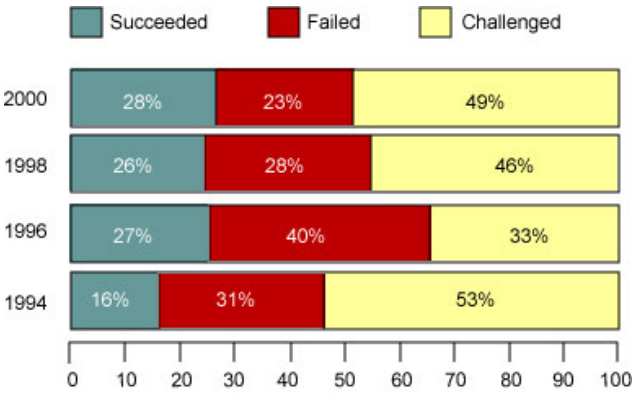
ですから、皆さんがまだそのように感じていないとしても、私のほうは、XP にすっかり夢中になっています。やがて皆さんもそうなると期待していますが、これは、巧みなマーケティングのため、というわけではありません。私は、他の人々を説得する最善の方法は、ありのままの XP を、欠点を含めてすべて見せて、XP 手法が今使用している手法よりも優れているかどうかを、判断してもらうことであると考えています。

ビジネスの問題

「XP の真髄」を発表してから、(IT の内部および外部で) 企業の管理者が直面する問題は変化していません。つまり、リーダーたちが IT の資産と可能性を利用して競争を優位に進めるにはどのようにすればよいのか、ということです。多くの場合、私たちの議論の対象となる「資産と可能性」は、ソフトウェアおよびソフトウェアの作成を中心としたものです。問題の原因はそこにあります。

プログラマーたちは、50 年以上にわたってコードを作ってきました。その間、コードが山ほど多く書かれました。そのうちよいものもありましたが、多くのものは質がよくありませんでした。コードの平均的な出来栄えが悪い原因は単純明快です。ソフトウェア開発のための伝統的な手法がプロジェクトを失敗させてきたためです。最も困ったことは、善良で勤勉で賢明な人々が、おびただしい数のプロジェクトを失敗させてきたことです。私たちは「XP の真髄」で、プロジェクトの成功数について話をしました。図 1 は、2000 Group CHAOS Report が報告した、新しい数値を示しています。(ついでですが、私は編集者から、「CHAOS」という用語を定義してはどうかと勧められました。私は、自分はよいのだが、Standish Group が頭字語の意味を明かすことを拒んでいると答え、「それに、君を巻き込みたくはないしね」と付け加えました。「Keeping CHAOS quiet」(参考文献を参照) で、その宣伝文句を読むことができます。)

図 1. ソフトウェア・プロジェクトの成功と失敗の推移



私は、これと似たグラフをソフトウェア関連雑誌の記事で目にしました。著者はこのグラフに「プロジェクトは着実な進歩を示している」という表題を付け、1994 年にはプロジェクトの 16 % だけしか成功していなかったのに対し、2000 年には 28 %が成功しているという注記を加えていました。確かに、28 %というのは 16 %よりはよい値ですが、それでも、惨めな結果です。前に述べたように、標準のソフトウェア開発手法を使用するのであれば、たとえ Java アプリケーションを開発する場合にも、あとで落胆することを覚悟しておく必要があります。1994 年に最初の CHAOS レポートが発表されてから、ある程度の改善が見られるとはいえ、プロジェクトのほぼ 4 分の 3 がまだ失敗しています。企業の指導者たちがソフトウェア・プロジェクトへの投資に気乗りがしないのは、無理ありません。

なぜ、このような低い数値になるのでしょうか?おそらく、この問題に関しては、問いを発した人の数だけ意見があると思いますが、私は、基本的な理由は次のようなものであると考えています。

- 人々が、問題が存在していることを納得していない。
- 問題の存在は認識しているが、それを解決するためにほかの方法を試みるのをためらっている。
- 問題の存在を認識していて、それを解決しようとする意思があるが、解決しようとしている問題について誤解している。
- 問題の存在を認識していて、それを解決しようとする意思があり、問題を理解しているが、現状を維持するように制約を受けている。

これらに理由について、簡単に説明することにします。

問題が存在することを納得していない

人間は、自分をだますのが非常に巧みです。企業やプロジェクトのリーダーも例外ではありません。万事順調であると皆が確信している (あるいは、少なくともそのように振る舞っている) にもかかわらず、ソフトウェア開発プロジェクトが企業にとって無駄な出費になることは、十分に考えられます。私が先ほど示した数字を見ると、組織におけるソフトウェア開発がうまくいっていないことは明らかです。もちろん例外はありますが、多くの組織は、自分たちが抱えている問題に気付いていません。

皆さんの組織では、IT と実務担当者との間に敵対関係はありませんか?あなたの組織のビジネス・リーダーたちは、「テクノロジーが巧妙なものなら、なぜ IT からはノーという答えしか返ってこ

ないのかね」などということを言っていませんか?もしそうであるとすると、解決しなければならない問題が存在していることになります。この問題を解決しないと、しばらくは惰性でなんとかなるとしても、やがて失敗に見舞われます。

問題の存在は認識しているが、それを解決するのをためらっている

より一般的に、組織内の知的で観察眼の鋭い人々は、現在使用しているソフトウェア開発方式が通用しないことを認識しています。彼らは、状況を改善するためにあえて別のことを試みることをためらっているのです。これは理解できます。新しいことを試みるには、勇気が必要であり、多くの場合には危険を冒す必要もあります。それに、素早く勝利を手にして、すぐに満足したいというわれわれの文化では、失敗は、自分の経歴を傷つける可能性があり、ことによっては致命傷ともなりかねません。

不利な状況では、ほとんどの人は最も抵抗の少ない道を選びます。この手法を取ると、短期的には経歴は安泰ですが、問題が先送りされ、複雑になるだけです。あまり長く待っていると、小さな問題が手におえないほど大きくなってしまいます。結局のところ、勇気を奮って思い切ったことをするように人々を仕向ける方法となるものはありませんが、このコラムの記事および関連のフォーラムに記載されるアイデアは(皆さん自身を含め) 組織内の人々が失敗への恐怖を乗り越える助けとなるのではないかと思います。

問題の存在を認識していて、それを解決しようとする意思があるが、解決しようとしている問題について誤解している

人々は、問題の存在を認識していて、たとえその解決のために新しいことを試さなければならないとしても、その解決に踏み切ろうとすることがあります。しかし、彼らが解決しようとする問題が誤って いることがあるのです。例えば彼らは、ソフトウェアの開発は製品を組み立てラインに載せるようなものであると (おそらくは無意識に) 想定して、生産の問題を解決しようとします。組み立てラインの作業員に、効率のよい組み立て順序および方法から外れたことを行わせるのは、理にかなうことではありません。ソフトウェアは、新しく現れた実体ですので、事情が異なります。XP2002 におけるプレゼンテーション Agile Methodologies: Problems, Principles, and Practices (参考文献を参照) の中で、Jim Highsmith は、ソフトウェアの最適化と探究の違いを説明しています。ソフトウェアに関しては、私たちは多くの場合新しい分野を探究し、これまで行われなかったことを行っています。したがって、ソフトウェア開発は、異なる解決策を必要とする、まったく別の種類の問題なのです。

ソフトウェア開発プロセスを過度に「機械化」あるいは制御しようとすると、制御が効かなくなってしまいます。皆さんが解決しようとしているソフトウェア開発の問題を明確にするために、このコラムの今後の記事が役立つことを願っています。

問題の存在を認識していて、それを解決しようとする意思があるが、現状を維持するように制約を受けている

最後に、問題の存在を認識していて、それを解決しようとする意思があり、解決すべき問題を理解しているにもかかわらず、組織内で身動きが取れない場合があります。これは、不幸で克服しがたい問題です (しかし、必ずしも克服できないわけではありません)。残念なことに、これを解決するには、大多数の人々が持ち合わせている以上の勇気が必要です。ときには、変化を促すために、人の機嫌を損ねなければならないことがあります。実際に行うとなると、口で言うほど簡

単ではないことは私にも分かりますが、所属する組織のリーダーが、組織の健全性を害しかねない問題への取り組みを拒んでいる場合には、皆さんは、現状を打破して変化を生み出すことを選ぶか、組織が崩壊する前に辞める道を選ぶか、決断を下す必要があります。今回の記事、およびこのコラムにおける今後の記事の内容は、皆さんの組織に存在する、新しいこと全般への (あるいは特に XP への) 抵抗を乗り越えるために役立つはずです。

1 つの解決策: アジャイル・メソッド

ある意味で、「エクストリーム・プログラミング」は名前で損をしています。大多数の人たちは、XP と聞いて、過激なスポーツ (extreme sports) あるいは Microsoft のオペレーティング・システムを思い浮かべるのではないかと思います。この名前の背後にあるのは、ソフトウェア開発のためには、単体テストを書くことやコード・レビューなどの、ベスト・プラクティスが存在するという考え方です。このプラクティスをいつも実行に移そうではありませんか?それらのことを行うと、それらがテスト主導型開発やペア・プログラミングなどのような、多くの人によって極端なものと考えられている概念に変化します。エクストリーム・プログラミングという名前の由来はそこから来ています。炭酸飲料や、バンジー・ジャンプ、あるいはビル・ゲイツとは関係ありません。

おそらく、この名前についての的外れで先見の明のない批判をかわすためと思われますが、XP などの手法に好意的な人々は、それらの方式をアジャイル と呼ぶようになってきました。Crystal Methods、Adaptive Software Development、および (現在最も人気の高い) XP がアジャイル・メソッド と呼ばれるのを耳にすることがあると思います。これらのプロセスはすべて、ソフトウェアを開発するために人と人とが交流するという、現実を受け入れています。ソフトウェア・プロセスを成功させるためには、人々の強みを最大化し、弱みを最小化しなければなりません。私の意見では、XP は、関係する人々に影響を与えるあらゆる補完的な力に対応し、チームに属するプログラマーたちが、コードを書くときに正しい作業が行えるように援助する点で、最高の機能を発揮します。またプログラマーたちは、多くの時間を、彼らが望むところのコードの作成に割くことができるようにもなります。

その方法をアジャイル と呼ぶのか、別の呼び方をするのかは、問題ではありません。結果が問題です。目標は、ソフトウェアの開発を合理的に行うために、すべての力のバランスを取ることです。私の意見では、この手法を実施するためには、常に厳しい率直さが要求されます。あきらめて旧来の方法に戻そうとする圧力は常に存在しますので、それに屈しないでやり抜く覚悟が必要です。もちろん技術も必要です。

多くの人の言うこととは逆に、私は、すべてのプログラマーが XP チームで本領を発揮できるとは考えていません。XP 手法を遂行するためには、常に強固な率直さと謙虚さが要求されます。プログラマーの多くは、そのようなことを学校で教わっておらず、またほとんどの企業環境では、そうした行動が報われることがありません。ソフトウェアは、人間が開発するものです。私たちはだれでも弱点を抱えていて、それらの多くを他人と共有しています。人を人間らしく扱ったうえで、彼らが誤った行動を取りにくくするような方法が必要です。私は XP が、これまで私が目にしたどの方法よりも、そうしたことをうまく行えると信じています。

XP の価値

「XP の真髄」で私たちが述べたように、XP は、ソフトウェア開発者がやるべきこと (つまり、コードの作成) を最もよく行えるような、価値とプラクティスの中核的なセットを規定していま

す。XP は、大掛かりなプロセスにありがちな、開発スタッフの能率を低下させ、彼らを消耗させる、目標から外れた不必要な作業 (ガント・チャート、状況報告、何冊もの必要書類など) をなくします。Kent Beck は、彼の著書 Extreme Programming Explained: Embrace Change (参考文献を参照) の中で、XP の中核的な価値について概説しています。これらの価値は、この一年の間でも何も変わっておらず、私なりに要約すると、次のようになります。

- コミュニケーション: プロジェクトの問題は、多くの場合、ある人が他の人に、なんらかの時点で、重要なことを話していないことに起因することがあります。XP は、コミュニケーションを行わない ことをほぼ不可能にします。
- 単純さ: XP は、プロセスおよびコード作成を行うにあたり、可能な限り単純なことを行うことを提案します。Kent はこの点をこう語っています。「XP はかけをしている。使われるかどうか分からないのに、今複雑なことをするよりも、指し当たっては単純なやり方で済ませたほうがよい . . . ということにかけているのである。」
- フィードバック: 顧客(customer)、チーム、および実際のエンド・ユーザーから具体的で素早いフィードバックが得られると、作業を「かじ取り」する機会が増えます。フィードバックのおかげで、道からそれて側溝に落ちる心配がなくなります。
- 勇気: 勇気は、他の 3 つの価値のコンテキストの中で存在します。これらの価値は相互にサポートし合っています。常に具体的なフィードバックが得られるほうが、前もってすべてを知っておこうとするよりもよいと信じるためには、勇気が必要です。自分の無知をさらけ出しかねない状況でチームの誰かに話をするには、勇気が必要です。明日決定すべきことを明日まで決めずにがまんして、システムを単純なものにとどめておくためには、勇気が必要です。また、単純なシステム、知識を広めるための絶え間ないコミュニケーション、およびかじ取りのためのフィードバックがなければ、勇気を持ち続けるのは困難です。

イタリアのサルディニアで開催された XP2001 コンファレンスのために共同執筆した報告書で、私は、このリストに「 内省」を付け加えるべきであると提案しました。しかし、そこで報告を行っているときに、実は内省は単なるプラクティス以上のものであることに気が付きました。もう 1 つの価値として付け加える対象となる、この強力な候補は、Joshua Kerievsky の論文 (参考文献を参照) を基に考え出されたものです。彼は同書の中で、健全な XP チームにとって継続的な学習がいかに必要であるのかを説いています。私も同感です。新しい価値をもう 1 つ追加することに、XP の実践者の間でどのような支持があるのかは分かりませんが、私は追加するに値するものであると思います。

改訂の必要性

私は、初めて「Extreme Programming Explained」を読んだときに、興奮のあまり、ほとんど一気に読み通してしまいました。私が目にしたことのある企業環境の多くと、Kent が述べる、誠実ですがすがしい手法との間の著しい違いに驚かされました。ただし私は、この本がプロジェクトのビジネス面に関して、想像に任せている部分が多すぎるのではないかというひそかな疑念を感じ、彼の言うことをそのまま受け入れるのに抵抗を覚えました。しかし、私は、XP が単なるプログラミングをはるかに超えたものであるという考え方に衝撃を受けました。

幸いにも、XP コミュニティー、インターネット・ニュースグループ、および XP の「12 のプラクティス」に関するその他のフォーラムで、かなりの議論が行われています。人々は、XP の最初のリリースで、ソフトウェア開発におけるプログラミングの側面が、他の事柄を犠牲にしてまでも重視されていて、ビジネス面の機能が、必要性を満たすほど完備していないことを知ってい

ます。基本原則は立派なのですが、完成の域に達するためには、プラクティスをさらに付け足す必要があります。改訂されたプラクティス・リスト (これには、12 か条ではなく 19 か条のプラクティスが含まれています) には、XP が実は単なるプログラミングを超えたものであることを認識することの必要性が、暗黙の形で含まれています。大掛かりなソフトウェアを作成するためには、ある意味で組織を変革することが必要であるというわけです。この手法を取るためには、見事なまでのコーディングの規律とスキルが要求されることは確かですが、同時に、チーム全員のソフトウェア作成に関する考え方を大幅に変化させることも必要です。つまり、ユーザー、ビジネス・アナリスト、ストラテジストを含む、チームの全員 の考え方を変える必要があります。新しいプラクティスは、この点をきわめて明確にしていて、すばらしいソフトウェアを永続的に作成する一体となったチームを作ることを重視しています。実際、一体となったチームをこのように重視することは、XP の世界で起こる変化の背後にある主要な原動力となっています。このテーマに関する「ワン・チーム」(One Team) という論文の草案で、Kent Beck は次のように述べています。

> 「Extreme Programming Explained」における最大の誤り は、単一の顧客を相手にしている技術チームというものを暗黙のうちに想定していることである。

同じ論文の後の部分では、次のように述べています。

> 「XP Explained」(およびその後の著作や会談) では、「顧客はチームに対して、ひとつのまとまった意見を話す」などの言い方の中で、「チーム」という用語をプログラマーを指すために使用したことにより、問題を複雑にしてしまった。このような言葉の使い方は常に私を悩ませたが、私には、どうすればよいのか分からなかった。

新しいプラクティスとワン・チームへの重視は、この懸念に対処しようとするものです。ソフトウェア開発に組織の変革を持ち込もうとすると、IT とビジネスの間の不和を解決しなければ ならなくなります。そもそも、不和が存在する最大の理由の 1 つは、ソフトウェアの作成にかかわる人々が同じチームをなしていなかったりするためです。それでは、ワン・チームの場合にはどうなるのでしょうか?同じ論文で、Kent は 3 本脚のいすの例えを使用しています。開発チームはプログラマーからなり、ビジネス (または顧客) チームはアナリスト、ユーザー、テスト担当者などからなります。3 本目の脚は管理者チームであり、これは次のような疑問に答える人たちによって構成されています。

- プロジェクトをどのように立ち上げるのか?
- 投資の増額、削減、あるいは打ち切りは、どのように行うのか?
- ビジネス・チームと開発チームによって対処できない意見の相違を、どのように解決するのか?
- このチームと、完了させる必要のあるその他すべてのプロジェクトとの間の、相対的な優先順位をどのように設定するのか?

このようにして、ワン・チームが、プログラマー、顧客、および管理者の 3 つをサブチームとして定義されることになります。

管理者チームが業務上の大きな決定を行いますが、一方、いずれかの単一のチームのみが開発を担当するわけではありません。すべての人が一体であるチームに所属し、各人が異なる役割を果

たします。もちろん、実際に行うのは、口で言うほど優しくはありません。大勢の人が、物事を異なったやり方で行おうとする意欲を持つ必要があります。失敗の危険 (さらに、場合によっては打ち切りの危険) がありますが、大きな見返りが得られる可能性があります。

なるほど、すばらしい。ワン・チームを作るとしましょう。でも、チームの皆は何をするのでしょう?1 年前と同じように、XP のすべてのプラクティスは、4 つの価値を基に、あなたが技術者であるかどうかを問わず、ソフトウェア開発チームの一員として毎日行うべきことを決めます。ここには、いずれかの役割に就いている人にとって、新しいことはあまりありません。XP プログラミングのプラクティスは、何年も前から業界ではベスト・プラクティスとして認識されてきました。そして、ビジネスに焦点を合わせた多くのプラクティスの効果が認められています (その証拠は、参考文献 に挙げた「Standish Group CHAOS Report」に示されています)。エクストリーム・プログラミングの「Extreme (極限)」という言葉が次の 2 つのことに由来していることは、依然として真実です。

- XP は業界で定評のあるベスト・プラクティスを採用し、最大限に活用している。
- XP はこれらのプラクティスを組み合わせて、それぞれの部分の単純合計よりも多くの成果を得られるようにしている。

全員が同じチームに所属したとすると、2 番目の成果は驚くべきものになる可能性があります。しかし、いすの脚のうちの 1 本を切り落とすと、たちまち床に落ちてしまいます。XP は、人々が常に正しく行動することを保証したことはなく、これからもないでしょうが、それを試みる機会を提供しています。これは、技術者であるかどうかにかかわらず、チームの全メンバーについて当てはまります。ワン・チームの人々がすべてのプラクティスを一緒に行うと、速度と効率が大幅に向上する可能性があります。これらのプラクティスにより、組織がすばらしい ソフトウェアを作成できるような環境を作ることが可能です。さて、そのほかに何か必要でしょうか?

改訂されたプラクティス

XP の 19 のプラクティス (表 1 を参照) では、チームのさまざまなメンバーが守るべき習慣を定義しています。今月は、共同のプラクティスについて詳しく採り上げます。これは、ワン・チームのすべてのメンバーをまとめるものであるからです。来月は、開発のプラクティスを取り上げる予定です。オリジナルの 12 の XP プラクティスの大部分は、これに該当します。それ以降は、顧客および管理者のプラクティスを採り上げる予定です。これにより、ソフトウェアを XP で開発することの意味がよく理解できるはずです。

表 1. XP の 19 のプラクティス

共同のプラクティス	反復 共通の用語 オープンな作業空間 回顧
開発のプラクティス	テスト主導型の開発 ペア・プログラミング リファクタリング 集団的な所有権 継続的インテグレーション YAGNI
管理者のプラクティス	責任の受け入れ

	援護 四半期ごとの見直し ミラー 持続可能なペース
顧客のプラクティス	ストーリーの作成 リリース計画 受け入れテスト 頻繁なリリース

共同のプラクティスについて論じる前に、プラクティスとはどのようなものであって、どのようなものでないのかを明確にさせてください。プラクティスだけでは、XP とはなりません。XP はプラクティス以上のものです。実は、XP の価値が XP そのものという訳でもありません。Ken Auer、Erik Meade、および Gareth Reeves は、「The Rules of XP」(参考文献を参照) という論文の中で、このことをうまく言い表しています。彼らの説明を要約すると、次のようになります。

> XP の価値は XP をアジャイルなものにする。XP のプラクティスが XP を定義するのではなく . . . XP はその規則によって定義される。

彼らは、Alistair Cockburn によるゲームとの類比を借用して、「参加(engagement)の規則」について説明しています。この規則は、ソフトウェア開発を効率よく行える環境を表すものです。その後で、「活動(play)の規則」について論じています。この規則は、参加の規則の枠組みの中で、分刻みの活動と規則を定義するものです。彼らは、次のようなことを言っています。

- XP をユニークなものにしているのは活動の規則である。
- エクストリーム・プログラミングは活動の規則に従ったものである。
- エクストリーム・ソフトウェア開発は活動の規則および参加の規則に従ったものである。

XP のプラクティスが重要なのは、あるチームについて、所属メンバーが望む限りすばらしいソフトウェアの作成を続ける機会を増やすような振る舞いを促すからです。しかし XP はそれだけにとどまるものではありません。単なるプログラミングの枠を超えて、ソフトウェア開発プロセス全体にかかわります。改訂された XP プラクティスは、これを考慮して、関係する全員に異なる振る舞いを要求しています。

XP のプラクティスが 4 つのカテゴリーに分けられることが、しだいに明らかになりつつあります。ワン・チーム内の各サブチームごとに 1 つずつプラクティスのセットがあり、3 つのサブチーム全体に適用されるプラクティスのセットがあります。オリジナルの 12 のプラクティスのうちのほとんどは開発チーム・セットに属しますが、そのうちのいくつかは、他のセットに移動するものと思われます。いくつかの名前は変更されました。これは主として、オリジナルの名前があまり適切でなかったか、あるいは思ったほど意味をなしていないためです。いくつかのプラクティスは、オリジナルでリストされた 12 項目には含まれていなかったものです。次のセクション (および今後の記事) では、それぞれの名前の後に続けて、そのプラクティスが新規のものであるのか、未変更であるのか、あるいはオリジナルの名前の中に対応するものがあるのかを注記することにします。これらの名前は流動的であっても、本質は流動的でないことが多いので、注意してください。

プラクティスについて説明する前に、私が「小刻みな議論」という言葉をどのような意味で使用しているのかを、はっきりさせておく必要があります。あるチームがすべてのプラクティスを使用する必要があるのでしょうか、あるいは、気に入ったものだけを選ぶことができるのでしょう

か?規律としての XP は、チームがすべての プラクティスを常に 使用することを想定していますが、チームは、いくつかのプラクティスを選択して、それらを首尾よく使用することができます (リファクタリングやペア・プログラミングの場合など)。皆さんが、自分のチームで、そのようにするのも自由です。私は、自分のチームではそうしないつもりです。プラクティスは相互に促し合い、一緒に行われることによってうまく機能するのであって、いずれか 1 つまたはそれ以上を除外してしまったのでは、すばらしい機能を利用できなくて損をすることになると思うからです。しかし、判断を下すのは皆さん方自身です。XP を試してみたい場合には、しばらくの間は常にすべてのプラクティスを使用して、どれは断念することになるのかを見極めることをお勧めします。きっと、その (断念することになるものの) リストは小さなものになるはずです。

さて、プラクティスに戻りましょう。今月は、他のすべての事項のための準備段階として、共同のプラクティスについてお話しします。最初に、すべてのプラクティスを一緒に使用する単一のチームが存在していることを想定する必要があります。これが前提条件となります。

共同のプラクティス: ワン・チームを作り上げる

このプラクティスのグループは、全員に適用されます。チームの全員 (プログラマー、顧客、および管理者) が、常にこれらを実行する必要があります。これらのプラクティスによってサブチームがまとめられ、そしてワン・チームが作成されます。

共通の用語 (メタファーに対応)

オリジナルのメタファー・プラクティスは役に立ちませんでした。アイデアは悪くありませんでした。全員が理解できる言葉でシステムを記述し、統一的なテーマをそのシステムに割り当てて、新しいことがどこに適用されるのかをチームが知ることができるようにするような、統制の役割を果たすイメージを使用するという考えです。問題は、このプラクティスが主要な目標である、システムへの理解を共有することからそれていたことです。それは、チームの全員にとって必要なことです。非常にうまいメタファーというのは、見つからなかったり、それを作り上げるのに時間がかかりすぎたりする場合があります。構成しようとしているシステムを一言でうまく言い表す言葉が見付からないというだけの理由で、時間を無駄にしてはなりません。全員が使用できるような用語 (Ward Cunningham の言う「名称のシステム」) を統一することに専心してください。メタファーが役に立つようであれば、それを使用してください。よいメタファーが見付からない場合には、共用される用語 (クラス、メソッド、その他の名前) に関する合意を図るだけでかまいません。

Bill Wake も「Extreme Programming Explored」(参考文献を参照) の中でメタファーについて語っています。彼は次のように言います。

> メタファーは探究段階で、つまりいくつもストーリーを書いて没にするような時期に決定してください。作業の進展とともに、それを改訂していってください。そのメタファーに従ってソリューションを考えるようにしてください。最上位のクラスには、そのメタファーの名前を使用してください。それらのクラスがどのように相互作用するのかを理解してください。

彼は、よりよいメタファーが思い浮かばないときには、「素朴なメタファー」を使用すること (つまり、オブジェクトをそのままにしておくこと) を勧めています。よいアドバイスと言えます。

オープンな作業空間 (新規)

非公式のコミュニケーションは、多くの場合、正式なコミュニケーションよりも効果的です。間仕切りの壁があると、非公式のコミュニケーションがやりにくくなります。最善の解決策は、作業空間をオープンにして、ほかの人の会話がお互いに聞けるようにし、役に立つ意見を提供し、重要な事項に関していつでも話し合いができるようにすることです。このようなグループの対話により、これまでにないようなすばらしい結果が得られることがあります。私も、実際にそのようなことを経験したことがあります。壁は取り壊してください。そうすることにより、チームはよくなります。皆さんの組織で、オフィス家具の移動が難しそうな場合、問題はもっと深刻だということです。

オープンな作業空間に対する典型的な反応は、同じエリアで何人もの人が話を交わすと、相互の能率が低下しないかという懐疑を持つことです。経験から言って、そのようなことも確かにあります。オープンな作業空間がうまく機能するためには、その作業空間でチームのメンバーが仕事をできるようにお互いを尊重し合うことが必要です。おしゃべりをしたりするのではありません。作業空間とは別に息抜きエリアを設けて、息抜きをしたい人は、そこに行くようにすればよいのです。だからと言って、作業空間が楽しいものであってはならないということではありません。作業空間は楽しいものにするべき です。しかし、同時にまた、仕事が達成される場所でなければなりません。チームのメンバーは相互に気をつかい合い、他人が行っている仕事に配慮し合うならば、オープンな作業空間はうまく機能するようになります。話を交わすことが問題であると決めてかからずに、問題となったときに対応するようにしてください。

私は、オープンな作業空間というプラクティスがあることは、XP が大規模なチームでは機能しないことを意味するという意見を聞いたことがあります。つまり、大勢の人が 1 つの広い部屋で仕事をすると、混乱に陥ってしまうというのです。多分そのとおりでしょう。しかしはっきり言って、これは極端な例です。こうした意見は、2 つのことを前提としているようです。大規模なチームが必要であるということと、チームを支える協調的な環境が存在しないということです。

大規模なチームが必要な場合もあるでしょう。しかし、本当に大規模なチームが必要だと考えたのでしょうか、チームが大きくなってしまっただけではないのでしょうか?おそらく、優れた開発者からなる小規模なグループが、大きな仕事にかかわる環境で (必要なサポートを得ながら) 仕事をすると、大規模なグループよりもより大きな仕事を、よりよく、短時間で仕上げることが可能でしょう。そうはならない可能性もありますが、考慮するだけの価値はあります。

大規模なチームがどうしても 必要であると仮定します。大混乱を引き起こさずに協調的な環境を作るには、どうすればよいのでしょうか?複数の作業空間が、不完全な障壁 (開いたままのドア、低い壁、衝い立てで仕切られた大きな空間同士の間にある通路など) で仕切られている場合はどうでしょうか?大規模なチームに適した協調的な環境を作ることはできないという人々の大半は、チームが小規模であろうが大規模であろうが、一般的にこうした考えが気に入っていないのだと思います。オープンな作業空間が大規模なチームの妨げになると決めてかかるのは誤りです。

反復 (新規)

XP を使用する人々は、プロジェクトの「リズム」ということをよく口にします。このリズムはさまざまなレベルで存在しますが、反復は本物の 鼓動です。1 ~ 3 週間ごとに、チームは新規機能を含む稼動可能なシステムを、要求した顧客に提示します。これは、かなり過激なアイデアで

す。人々は、このような方法でソフトウェアを作成することに慣れていません。ほとんどの手法では、コードを書く前にいろいろなことをすることに懸命になります。XP では、コードを書き、それを人に使用してもらい、フィードバックに従ってプロジェクトをかじ取りすることが要求されます (この概念については、今後の記事でさらに説明します)。

それぞれの反復は、なんらかの計画で始まります。私 (および他の多くの人々) はこれを反復計画と呼び、反復計画会議 で実行しています。この会議で決められる計画は、リリース計画 (この概念についても、今後の記事でさらに詳しく説明する予定です) によく似ていますが、それよりも詳細なレベルで行われます。計画の際には、次の一反復のことだけを考えます。顧客に対して「プロジェクトが次回の反復で終了するとしたら、どのような機能を追加してほしいですか?」と尋ねます。そして、それらの機能を構築するための作業を見積もります。その反復の中で可能なことを行い、それ以外は後回しにします。顧客が優先順位を決め、プログラマーがコストを指定します。つまり、現行の反復には組み込むのをあきらめなければならないことも出てきます。これについては、妥協は許されません。時間を作り出すことはできないのです。

ここでは、タイムボクシングということが重要になります。タイムボクシング とは、ある分量の作業に時間制限を設け、その時間が終了するまで作業を行ったうえで、現状を評価する、というプラクティスです。これは、チームがタスクの作業に時間をかけすぎて、行う必要のある別のことができなくなってしまうことを避けるための技法です。反復プラクティスには、タイムボクシングにはないそれ以上の要素も含まれます。反復プラクティスの考え方は、困難な決定を、それに最もふさわしい人に行わせようということなのです。顧客は、次回に構築すべき最も重要な機能を決定します。プログラマーは、機能の構築方法、およびそのために必要な作業量に関する、技術的な決定を行います。管理者は、プロジェクトの戦略的な方向付け、および組織の全体像にそのプロジェクトを適合させる方法を決定します。このように、プロジェクト・チームに属する全員が参加しますが、その対象は自分が分かることだけに限定されます。各人は、自分の頭脳を使うことはもちろんですが、他の人がそれぞれの頭脳を使うことにも信頼を寄せることになります。

このプラクティスをどのような名前で呼ぶのかについては、議論がありました。反復の短さを強調して短期反復 と呼ぶべきなのか、あるいは、全員が計画に参加するというコミュニティーの側面を強調して反復計画 と呼ぶべきなのか、ということです。私は短期反復 という呼び方が気に入っています。具体的なフィードバックを得るために、実際に稼動するソフトウェアをユーザーの手に渡す必要があることが強調されるからです。

回顧 (新規)

私の経歴では、プロジェクトにおいてチームの全員が短い時間を割いて仕事を振り返り、まずかった点をオープンに指摘するということは、めったにありませんでした。こうしたことは、自分ひとりで行わなければなりませんでした。(時間があって) 見直しをした場合に、私が発見したプロジェクトの問題点の多くは、チームがほんの少しの時間を割いて対処しておけば、たいていは問題となる前に解決できたようなことでした、回顧 (Norm Kerth が考え出した用語です) とはそういうことです。

自分が行っていることを振り返らずに作業を行うと、いつまでも失敗を繰り返すことになります。チームの各メンバーが自分の行動および自分が学んだことを振り返ったうえで、その結果を他のメンバーと共有するようにしたほうがよいでしょう。Chris Collins と私が XP2001 で発表した

論文「Adaptation: XP Style」(参考文献を参照) で述べたように、私は個人の振る舞いを振り返ることを内省 と呼んでいます。回顧では、皆の前で内省を行うことになります。

プロジェクトにおいて学んだ教訓について考えてください。うまく行ったことや失敗したことについて考えてください。チームがどのように機能したのかを考えてください。チームが集団で作り出したものについて考えてください。もう少しうまいやり方はなかったのでしょうか?なぜですか? (このほうがより大切な質問ですが) どうすればよかったのですか?うまくできたので、これからも続けて行いたいと思うのは、どのようなことですか?それはどのように行ったのですか?こうした総体的な検討を、少なくとも各リリースの後で、できればそれぞれの反復の後で、組織的に行ってください。長い時間をかける必要はありません。反復計画における最初の 1 時間または 2 時間を、前回の反復の回顧に割り振ってください。リリース後の回顧には、おそらく 1 日 (あるいはそれ以上) を確保してください。回顧では、手加減はしないでください。チーム力を向上させ、組織に追加できる価値を増大させ、自分の人生をよりよいものにしてそれを仕事に還元しようとしているのですから。これはきわめて重要なことなのです。

私と、私の同僚のほとんどは、チームの各ペアにとって、一日の終わりに「ミニ回顧」を行うことが役に立つということに気付きました。ほんの数分間反省するだけでよいのです。私たちは、他のメンバーと共有するだけの価値があることを学んだときには、翌日のチーム・ミーティングで披露するようにしています。こうして、多くのことを学ぶことができました。

回顧が効果を発揮するためには、ある程度傷つけ合うことが必要です。その補償として、チームが一層統合されるわけです。技術関係の人も、それ以外の人も、全員が回顧に参加する必要があります。そのために、あまりにも多くの人が一か所に集まらなければならないようであれば (そのようなことは、十分にありえます)、サブチーム (管理者など) が独自に回顧を行い、より小規模なミーティングに代表を送るという方法もあります。重要なことは、学習によって、同じ過ちを何度も繰り返す事態を減らすことなのです。これはワン・チームである以上、全員が深くかかわる必要があります。

組織の変革

ソフトウェアからスタートした XP は、究極的には組織の変革を伴います。全員を同じチームに所属させ、しかも異なった仕事を与えるということは、私が想像するかぎり、よい変化を生じさせるための最良の方法です。今月は、こうした現実的で持続的な組織の変化を生むために必要なワン・チームを作るのに役立つ、XP の共同プラクティスについて検討しました。

今後数か月にわたる毎回の記事の終わりでも繰り返して言うことになると思いますが、ここで私が述べた改訂済みのプラクティスは、必ずしも「新規」XP を反映したものではありません。次期リリースがどのようになるのかを知るためには、Kent Beck による「Extreme Programming Explained: Embrace Change」の第 2 版を待つ必要があると思います。

ビジネスと IT のギャップを埋める方法

ビジネスのために必要なことを、実際にビジネスで使用されるソフトウェアに変えることは、IT の世界における最も困難な課題の 1 つです。技術畑の人間とビジネス畑の人間に絶えず十分なコミュニケーションを取らせるいう難題を前にすると、腰が引けてしまいます。ほとんどの企業の振る舞いを見ると、リップ・サービスとは裏腹に、本音ではコミュニケーションというものに価

値を認めていないようです。コミュニケーションの妨げとなる障害をいくつも設けています。情報の流れは滞り、行うべきこととは別のところで縄張り争いが行われています。他のグループが何をしているのか、まるで分かっていません。プロジェクトは開始までいつまでも待たされ、永遠の道のりを歩むような緩慢な足取りで進められ、しかもその多くは実を結ぶことなく立ち消えとなってしまいます。

皆さんが記憶しているかぎりでは、こうした状況が続いていたことでしょう。こうした振る舞いは習慣化されてしまいました。ビジネス畑の人は、技術畑の人からさまざまな形で「ノー」と言われるのに慣れてしまったため、新規プロジェクトに取り組むときには、可能なあらゆる機能への要望を詰め込み、たとえ事実と異なっていても、それらすべてが最優先機能であると主張するようになりました。この手のゲームにかけては、技術畑の人間のほうが年季がいっています。彼らには、ビジネス畑の人々がそれらの機能すべてを必要としているのでないことが分かっています。彼らはひそかに、ビジネス畑の人間のほとんどが愚かで決断力を欠いていると考えています。彼らには、ビジネス畑の人間が (おそらくゲームの後半で) 要件の変更を試みることが分かっていて、こうした「膨張」を許すと、ビジネス畑で必要としているものに近いものを、決められた時間と予算で提供する能力が損なわれることが分かっているのです。そのため、新規プロジェクトに取り組む際には範囲を限定することを試み、プロジェクトの実行に当たっては膨張を制限しようとします。このシナリオは、次のようなことを意味します。

- ビジネス畑の人は、IT は「ノー」という以外の言葉を知らず、自分たちは必要としているソフトウェアを手にすることができないと考えています。
- 技術畑の人は、自分たちが酷使されていると感じていて、作りたいと願っているソフトウェアを作ることができないでいます。

このゲームはそれ自体が問題ですが、他の問題の兆候でもあります。大もとの原因は人間性です。技術畑の人がビジネス畑の人とコミュニケーションを取らない本当の理由は、ビジネスの問題を解決しようと思っていないことが多いからです。そのようなことをするよりは、しゃれたおもちゃで遊んでいたいのです。そして、テクニカル・マネージメントの場面では、権力争いに勝利したいと考えています。もちろんこれは、すべての場合に当てはまるわけではありませんが、一般的であることは確かです。

ビジネス畑の人がコミュニケーションを取らない本当の理由は、余分な労力を注ぎ込んでまで、技術陣にきちんとしたことをさせようと思わないからであり、また、プロジェクトの結果への責任を負いたくないからです。ビジネス畑の人が、自分たちが本当に必要としているものやその理由を技術畑の人に理解してほしいと考えるのであれば、まず始めに自分たち自身がそれを理解する必要があります。自分でも分かっていないことが多いのです。ビジネス畑の人が、技術畑の人に「きちんとした」ソフトウェアを作ってほしいと考えるのであれば、そうなるように、技術畑の人と一緒に過ごす時間を割く必要があります。多くの場合、彼らはそうしたがりません (あるいは、そうすることが許されません)。ビジネス畑の人の多くは、そういうことが自分の仕事の一部であるとは考えていないのです。

こうした争いがあることを考えれば、ほとんどのソフトウェア・プロジェクトから得られる結果が芳しくないことは、さして驚くべきことではありません。むしろ、人々がこのように行動するような環境で、実際に成功するプロジェクトが多少とも 存在することのほうが驚きです。それでは、この現状をどのように修正すればよいのでしょう?ここで必要となるのが、組織のオーバーホールです。漸進的な変革では、おそらく現状は打開できないでしょう。(ただし、そのような方

法による変革を試みたいという組織があれば、お手伝いしたいとは思っています。)こうした変革の主要なきっかけとなるのは、ソフトウェア開発チーム内のビジネス畑のメンバーと技術畑のメンバーの協力を要求し、またそれを可能にする、これまでと大幅に異なるソフトウェア開発方法です。この役割を担うのは、ほかならぬ XP であると、私は考えます。

私が前に述べた、ワン・チームの概念を思い出してください。もしも皆さんが技術畑の人である場合、一般に、ビジネス畑の人を自分のチームの一員と考えますか?もしも皆さんがビジネス畑の人である場合、自分をソフトウェア開発チームの一員と考えますか?ソフトウェアの作成方法を劇的に変えるためには、人と人とのかかわり合い方を変える必要があります。XP では、それが要求されます。また、このコラムの今後の記事で述べる顧客プラクティスは、この点を非常に明確にしてくれます。顧客は、プロセスを開始から終了にいたるまで推進する必要があります。プログラマーである私にとって、そうした顧客は私のチームのメンバーです。わたしたちは、チームの中では一緒なのです。同じように、ビジネス畑の人も、プロジェクトを積極的に推進する必要があります。彼らは、どの機能を組み込み、どの機能を外すかという、困難なビジネス上の決断を下す必要があります。たとえシステムが「完成」していなくても、ソフトウェアを早期に、しかも頻繁にリリースして、実際の顧客から具体的なフィードバックが得られるようにする必要があります。特定のストーリーが「完結」したことを、顧客テストの形で示す必要があります。ラジカルな変革とは、そういうことです。

XP は単なるプログラミングをはるかに超えたものです。XP は、ラジカルで、生活を変えるような、組織変革を伴います。私は、企業の IT にはびこる病気の治療法は、XP 以外にはないと信じています。

来月の予告

今月のコラムでは、いさかいの絶えないグループではなく ワン・チームを作り上げるために役立つ、XP の共同プラクティスの概要を述べました。来月は、プログラマーのためのプラクティスについてお話しする予定です。これらのプラクティスはプログラマーに対し、皆さんのワン・チームが必要とするシステムを作るために、日常的に行うべきことを示します。ペア・プログラミングと聞いて、奇異に思ったり、ばかげていると感じたりしませんか?継続的なインテグレーションなど不可能であるとは思いませんか?来月は、それらの意味について説明し、それらが皆さんが考えているほどおかしなものでないことを分かっていただきたいと思います。

著者について

Roy Miller

Roy W. Miller は最初に Andersen Consulting (現在の Accenture) にて、10 年以上技術コンサルタント、ソフトウェア開発者、および指南役を務め、ノースキャロライナの RoleModel Software 社にてほぼ 3 年を過ごしました(ここでは彼は、エクストリームプログラミング (XP) を使用した Java 言語アプリケーションの構築に注力しました)。現在、彼は独立コンサルタントであり指南役です。彼は、XP を含む重要なメソッドおよび機動的なメソッドを使用しており、Addison-Wesley XP Series (Extreme Programming Applied: Playing to Win) 中で本を共同執筆しました。彼の最も最近の本である Managing Software for Growth: Without Fear, Control, and the Manufacturing Mindset は、どのように、複雑な技術がソフトウェア開発を支援することができ、そして他の IT マネージャー達が、プログラマをコントロールしたり枯らすことなく、彼らのチームが実際の人々が使用して楽しいような素晴らしいソフトウェアを作成することの、支援方法を理解するかを述べています。