小規模プロジェクトでの Rational Unified Process の使用: エクストリーム・プログラミングからの発展

Gary Pollice

Rational Software ホワイト・ペーパー

TP 183, 3/01

Rational®
the software development company

目次

要約

Rational Unified Process® または RUP® 製品は、標準のインスタンスを備えた完全なソフトウェア開発プロセス・フレームワークです。RUP から派生するプロセスは、製品サイクルの短い小規模なプロジェクトのニーズに対応する軽量プロセスから、大規模な、場合によっては分散されたプロジェクト・チームの幅広いニーズに対応する包括的なプロセスまでさまざまです。あらゆるタイプと規模のプロジェクトがうまく RUP を使用しています。このホワイト・ペーパーは軽量式の RUP を小規模なプロジェクトに適用する方法を説明します。また、エクストリーム・プログラミング (XP) テクニックを完全なプロジェクトの幅広い状況の中で適用する方法を説明します。

概要

ある実話

ある朝、上司に「今度会社が始める新規事業のために、簡単な情報システムを数週間でセットアップできないか。」と言われました。当時のプロジェクトに退屈を感じ、新しいことにあこがれていた私は、すぐにこの話に飛びつきました。 勤めていた大会社の官僚主義や手続きから解放され、機敏にすぐれた新規ソリューションを開発できると思ったのです。

出だしは好調でした。最初の 6 カ月間はほとんど自分一人で作業をしていました。長く楽しい期間でした。生産性は信じられないほど高く、生涯で最高の出来といえるものでした。開発サイクルは速く、ほぼ 2、3 週間ごとにシステムの主要な部分を新しく作成しました。ユーザーとの反復は単純かつ直接的でした。誰もが親密なチームの一員で、格式ばったことや書類を使うことなく意思疎通することができたのです。設計についても型にはまったものはほとんどありませんでした。言ってみれば、コードは設計であり、設計はコードでした。万事順調でした。

しばらくは何もかもうまくいきました。そのうち、システムが発達するにつれ、仕事も増えてきました。問題が変わると既存コードも進化せざるをえず、対応の必要性を認識するようになりました。そこで、開発の手助けをしてくれる人を数名雇うことにしました。私たちはチームとして活動し、問題の各部分についてペアになって作業することが多くなりました。意思疎通を積極的に行い、形式的なことはせずに済ませていました。

1 年が過ぎました。

人は増えつづけました。チームは 3 人から 5 人、やがて 7 人になりました。新しい人が入ってくるたびに長々とした習熟曲線が伸び、経験の恩恵を得ることなく、システム全体や概要さえも理解し、説明することもままならない状態になってきました。ホワイトボードに描いたシステムの全体構造図を取り込むことから初め、主な概念とインターフェースはもっと形式を考慮しました。

システムが必要な動作をするかどうかを検証するための主要な手段としてテストを依然使用していました。多くの新しい人が「ユーザー」側にいることから、プロジェクトの初期に通用していた型にはまらない要求や個人的な関係ではもはや対処できないと気付きました。構築しようとしているものを理解するのに多くの時間がかかるようになりました。そのため、議論の記録を取り、いつも結論を思い出すようなはめにならないようにしました。また、要求と操作シナリオを記述しておくことがシステムの新規ユーザーの教育に役立つことがわかりました。

システムの規模と複雑性が大きくなるにつれて、予想もしないことが起こりました。 システムのアーキテクチャーは意識的な注意を必要としたのです。開発初期の頃、アーキテクチャーはほとんど私の頭の中にあり、その後はノートに走り書きしたり、フリップ・チャートに描いたりしていました。しかし、プロジェクトに関わる人の数が多くなり、アーキテクチャーを管理するのが困難になってきました。私と同じ経歴をもった人ばかりではないので、アーキテクチャーに加えられた特定の変更を何となく察知することはできなかったのです。私たちはシステムにおけるアーキテクチャー上の制約をもっと正確な用語で定義しなければなりませんでした。アーキテクチャーに影響を与える可能性があるすべての変更はチームの合意と最終的な私の承認を必要としました。このことを発見するのは大変でした。アーキテクチャーの重要性を本当に認識するまでにいくつかの困難な教訓を得ました。

以上は実際にあった話です。ただし、このプロジェクトの苦い経験のほんの一部しか表していません。ただ 1 つ、この経験が珍しいのは、チームの数名が数年に渡ってプロジェクトの始めから終わりまで関わっていたことです。たいていの場合、プロジェクトのメンバーは入れ替わり、自分の活動が後のプロセスに与える影響を見ることがありません。

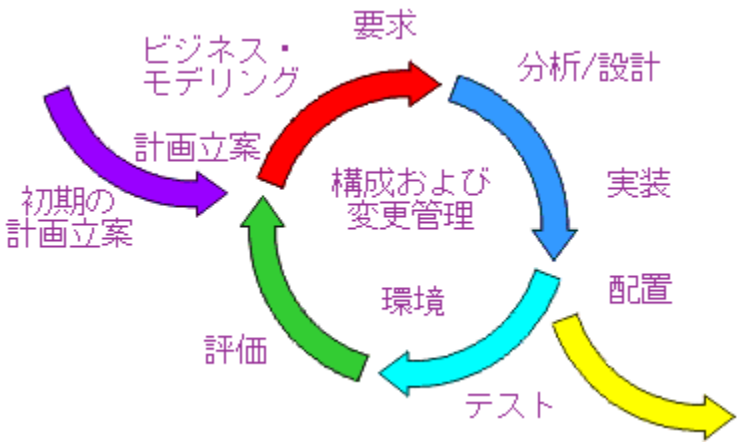
このプロジェクトはちょっとしたプロセスの工夫でもっとうまく進めることができたと思います。プロセスがあまりに多いと障害になりますが、プロセスが欠けていても新たなリスクが発生します。高いリターンのみを求めて高いリスクの株に投資する人のように、プロジェクト環境の重要なリスクを無視してプロセスをほとんど使用しないグループは、「最高の結果を望むが最悪の事態に準備しない」ことになります。

概説

このホワイト・ペーパーは前に述べたようなプロジェクトにプロセスを適用する方法を説明します。焦点を当てるのはプロセスの「適正レベル」を定めることです。プロセスを運用する開発チームとビジネス環境が直面する課題を理解することで、プロセス形式の適正レベルを判断することができます。こうした課題を理解した後、リスクを軽減するために必要十分なプロセスを供給します。軽量であれ何であれ、フリー・サイズのプロセスは存在しません。以降のセクションで、プロセスの適正レベルはリスクの機能であるという概念を考察します。

人気がある 2 つの手法を使用して、プロセスの適正レベルの特定方法について説明します。1 つは Rational Unified Process (RUP) 、もう 1 つはエクストリーム・プログラミング (XP) です。小規模なプロジェクト用に RUP をカスタマイズする方法と、XP が考慮しない領域を RUP で対応するしくみを紹介します。RUP と XP の組み合わせは、プロジェクト・チームがリスクを軽減し、ソフトウェア製品の出荷目標を達成するために必要なガイダンスを与えます。

RUP は Rational により開発されたプロセス・フレームワークです。RUP は業界で実証された 6 つの最善の実践原則に基づく反復開発手法です (RUP に関する付録を参照)。時間の経過と共に、RUP ベースのプロジェクトは、方向づけ、推敲、作成、移行の 4 つのフェーズを通ります。各フェーズには 1 つ以上の反復が含まれます。各反復において、要求、分析と設計、テストといった複数の作業分野のそれぞれにさまざまな量の労力を費やします。RUP の主要な推進力はリスク軽減です。RUP は、何千というプロジェクトで、何千もの Rational の顧客とパートナーによって使用され、洗練されてきました。下の図は一般的な反復のフローを示しています。



一般的な反復フロー

リスクがプロセスを形作る可能性がある例として、ビジネスのモデリングを行うべきかどうかについて考えましょう。ビジネスの理解の欠如が間違ったシステムの構築を生むという重大なリスクが存在する場合は、おそらく一定量のビジネス・モデリングを行わなければなりません。モデリング作業について形式を重視する必要があるでしょうか。それはユーザーに依存します。つまり、小規模なチームが略式に結果を使用するのなら、くだけたメモ程度のもので済むかもしれません。組

織のほかの人が結果を使用したり、レビューしたりする場合は、労力を追加投資して、提出物の正確性や理解しやすさを検討しなければならないでしょう。

RUP をカスタマイズしてほとんどすべてのプロジェクトに合わせることができます。どの標準プロセスまたはロードマップも特定のニーズに合わない場合、独自のロードマップを簡単に生成できます。ロードマップはプロジェクトがプロセスの使用を計画するしくみを説明します。したがって、ロードマップはプロジェクトの特定のプロセス・インスタンスを表します。これは RUP が必要に応じて軽くも重くもなり得ることを意味し、このホワイト・ペーパーで具体的に説明します。

XP は小規模プロジェクト向けのコード中心の軽量プロセスです (XP に関する付録参照)。XP は Kent Beck 氏により創案され、1997 年頃に Chrysler Corporation で行われた C3 給与プロジェクトでソフトウェア業界の注目を浴びました。RUP と同様に、XP は小規模リリース、シンプル・デザイン、テスト、継続的統合といったいくつかの実践原則を具体化する反復に基づいています。XP は適切なプロジェクトと環境に効果的ないくつかのテクニックを推進しますが、隠れた前提としてアクティビティーとロールが存在します。

RUP と XP は異なる理念から生まれたものです。RUP は、任意の特定ソフトウェア・プロジェクトに適用可能なプロセス・コンポーネント、手法、テクニックのフレームワークであり、ユーザーは RUP の専門家であることが期待されます。他方、XP は、完全な開発プロジェクトに適合させるには何らかの補足を必要とする制約されたプロセスです。こうした違いはソフトウェア開発の現場全体の認識に現れています。大規模なシステム開発者は RUP を問題の答えとし、小規模なシステム開発者は XP を問題解決策と考えます。私たちの経験からすると、ほとんどのソフトウェア・プロジェクトはそれらの中間にあり、それぞれの状況に合った適正レベルのプロセスを必要としているようです。どちらか一方だけでは十分とはいえません。

RUP の幅広さと XP のテクニックを組み合わせると、プロジェクトのすべてのメンバーが納得し、すべての主要なプロジェクト・リスクに対応する適度な量のプロセスを獲得できます。ユーザーがチームの一部になっている比較的信頼性の高い環境で作業する小規模なプロジェクトでは、XP がうまく機能します。チームが分散し、コード・ベースが大きくなったり、アーキテクチャーが十分に定義されていなかったりする場合は、何かほかの対策が必要になります。ユーザー対話について「契約で保証された」形態を実現しなければならないプロジェクトでは XP 以上の機能が必要です。RUP は、必要な場合により強固なテクニックで XP を拡張できるフレームワークです。

以降、このホワイト・ペーパーでは RUP の 4 つのフェーズに基づく小規模なプロセスを説明します。それぞれにおいて、生成されるアクティビティーと成果物を特定します。[fn]RUP と XP は異なるロールを責務を識別しますが、これらの違いについては言及しません。どんな組織またはプロジェクトでも、実際のプロジェクト・メンバーはプロセス内の適切なロールと関連付けられなければなりません。

プロジェクト開始 — 方向づけ

方向づけは新しい開発作業にとって重要であり、プロジェクトを進める前に重要なビジネスと要求のリスクに対応しなければなりません。既存システムを拡張するプロジェクトでは、方向づけフェーズは短いフェーズになります。 ただし、新規のシステム開発と同様に、プロジェクトを実施する価値があること、また、実現可能であることを確認することが重要です。

方向づけの間に、ソフトウェア構築の開発企画書を作成します。開発構想書は方向づけの間に生成される主要な成果物です。これはシステムの高レベルの説明です。開発構想書は、あらゆる人にシステムの内容を伝えるもので、使用するユーザー、使用する目的、提供される機能、制約事項などを示す場合もあります。開発構想書は、ごく短くすることが可能で、1 ～ 2 項の場合もあります。開発構想書は、ソフトウェアが顧客に提供しなければならない重要な機能を含むことがよくあります。

次の例は、Rational の外部向け Web サイトを再構築するプロジェクト用に描かれた非常に短い開発構想書を示しています。

> *訪問者セルフサービス、サポート、的確なコンテンツを提供する動的でパーソナライズされた Web 環境を通じて、顧客との関係を拡張することにより、e ビジネス開発分野 (ツール、サービス、実践原則) における世界的なリーダーとして Rational の位置付けを強化すること。新規プロセスと実現可能なテクノロジーにより、コンテンツ提供*

者は簡単で自動化されたソリューションを利用して発行を迅速化し、コンテンツの品質を向上させることができます。

RUP で指定される 4 つの基本的な方向づけアクティビティーは以下のとおりです。

- **プロジェクトの範囲を明確にする** — システムを構築しようとする場合、それがどういうもので、どのように利害関係者を満足させるのかを理解する必要があります。このアクティビティーでは、状況と最も重要な要求を、製品の受け入れ基準を導くことができる程度の詳細さで把握します。
- **開発企画を計画し、準備する** — 開発構想を指針として、リスク軽減戦略の定義、初期プロジェクト計画の作成、既知の費用、スケジュール、収益性のバランスを識別します。
- **候補となるアーキテクチャーを統合する** — 考慮中のシステムが目新しいものではなく、十分に理解されたアーキテクチャーが存在する場合は、このステップを省略できます。顧客の要求を理解したら、できるだけ早く、可能性があるアーキテクチャーの候補を調査する時間を割り振ります。新しいテクノロジーは、ソフトウェアの問題に対する新しく、改善されたソリューションの可能性をもたらします。プロセスの初期に時間をかけて購買と構築のトレードオフを評価すると同時に、テクノロジーを選択し、場合によっては初期プロトタイプを開発することで、プロジェクトの主要なリスクを減らすことができます。
- **プロジェクト環境を準備する** — どんなプロジェクトにもプロジェクト環境が必要です。ペア・プログラミングなどの XP テクニックを利用するにせよ、従来型のテクニックを利用するにせよ、物理的なリソース、ソフトウェア・ツール、チームが従う手続きを定める必要があります。

小規模なプロジェクトで方向づけを実行するのに多くの「プロセス時間」はかかりません。2 ～ 3 日中に方向づけを完了できる場合がほとんどです。以下のセクションでは、このフェーズで期待される開発構想書を除くそのほかの成果物を説明します。

承認された開発企画書

利害関係者は、ビジネスの観点からプロジェクトが実行に値することに同意する機会を持ちます。見込みのないプロジェクトに貴重なリソースを費やすのであれば、プロジェクトを行わない方がよいと早い時期に判断することが重要だという点で、RUP と XP は一致しています。「*Planning Extreme Programming*」(邦訳:「*XP エクストリーム・プログラミング実行計画*」)[1]に述べられているように、XP はプロジェクトが開始するしくみと関連するロールについてあいまいです (既存のビジネスまたはシステムの状況で最も明瞭のようです) が、探索フェーズで、XP は RUP の方向づけと同様の成果物を扱います。

XP のようにビジネスの問題を略式に考慮するにしても、RUP のように開発企画書を最高級のプロジェクト成果物にするにしても、それらを検討する必要があることに変わりありません。

リスク・リスト

リスク・リストは、プロジェクト全体を通して保守されます。リスク・リストは、リスクの簡単なリストに、計画された軽減戦略を付けたものになることがあります。リスクは優先順位が付けられます。プロジェクトに関係するすべての人はリスクの内容と対応法をいつでも確認できます。Kent Beck 氏は XP が対応するリスクのセットと対応方法を説明していますが、リスクの一般的な処理方法は提供されていません[2]。

---

[1] これは現在出版中の 3 つのエクストリーム・プログラミング関連書籍の 1 つです。

[2] 「*eXtreme Programming eXplained*」(邦訳:「*XP エクストリーム・プログラミング入門*」) の中で、Kent Beck 氏は 8 つのリスクと XP が対応する方法を説明しています (pp. 3-5)。読者はこれを参照して、十分であるかどうか判断することをお勧めします。私たちはほかにも多くのリスクが存在し、一般的なリスク処理戦略がすべてのプロセスに必要な部分であると考えます。

初期段階のプロジェクト計画書

この計画書にはリソース見積もり、開発範囲、フェーズ計画が含まれます。すべてのプロジェクトでこれらの見積もりは継続的に変化するため、監視しなければなりません。

プロジェクト検収計画書

正式な計画書を用意するかどうかはプロジェクトのタイプによって異なります。顧客がプロジェクトの成功を評価する方法を定めなければなりません。XP プロジェクトの場合は、顧客が作成する検収テストという形態をとります。より一般的なプロセスでは、顧客は実際にテストを作成しないことがありますが、検収の基準は顧客により直接、または顧客と対話するシステム分析者などの別のロールにより定められなければなりません。そのほかの検収基準として、エンド・ユーザー・マニュアルとヘルプの作成などが考えられますが、これらは XP で取り上げられていません。

初期の推敲反復についての計画書

RUP ベースのプロジェクトの場合、各反復の詳細を前の反復の終わりに計画します。反復が終了する際、開始時に指定された基準に対して進捗を評価します。XP は反復の成功を監視し、測定するためのテクニックを提供します。メトリックスは単純で、反復計画と評価基準に容易に組み入れることができます。

推敲

推敲フェーズの目的は、作成フェーズで行われる設計と実装作業の大部分に安定した基盤を提供するために、システムのアーキテクチャーのベースラインを確立することです。アーキテクチャーは、最も重要な要求の考慮 (システムのアーキテクチャーに強い影響を与えるもの) とリスク評価を通じて発展します。1 つ以上のアーキテクチャー・プロトタイプを通じて、アーキテクチャーの安定性を評価します。

RUP において、設計アクティビティーはシステム・アーキテクチャーの概念に焦点を当てます。 これは、ソフトウェア集約システムの場合はソフトウェア・アーキテクチャーに焦点を当てるということです。コンポーネント・アーキテクチャーの使用は RUP に統合されている、ソフトウェア開発における 6 つの最善の実践原則の 1 つで、アーキテクチャーの開発と保守に時間をかけることを推奨します。この作業に費やされる時間は、脆弱で硬直したシステムに関連するリスクを軽減します。

XP はアーキテクチャーの概念を「比喩」で置き換えています。比喩はアーキテクチャーの一部を捉え、アーキテクチャーの残りの部分はコード開発の自然な結果として発達します。XP は、最も単純な設計を作成し、コードを継続的に再構成することからアーキテクチャーが出現すると仮定しています。

RUP の場合、アーキテクチャーは比喩以上のものです。推敲の間、実行可能なアーキテクチャーを作成します。これにより、パフォーマンス、信頼性、堅固性といった機能外要求の充足に関連するリスクの多くを減らすことができます。XP の文献を読むと、RUP が推敲のために規定するもの、具体的には XP における基盤に関する過度の集中が無駄な作業であると推察できます。XP では、必要となる前にあらかじめ基盤を構築することに費やす労力は、非常に複雑なソリューションを生むことになり、結果として顧客にとって価値がないとされます。RUP において、アーキテクチャーと基盤は同じものではありません。

アーキテクチャーに対する手法は RUP と XP の間でかなり隔たりがあります。時間の経過と共に広がる開発範囲、追加されるプロジェクトの規模、新しいテクノロジーの追加に関連するリスクを回避するために、RUP はアーキテクチャーに注意を払うことを推奨します。XP は、既存のアーキテクチャー、またはアーキテクチャーがコード作成と共に発展可能なほど単純かつ理解可能なものであると仮定します。XP は明日のために設計することはせず、今日のために実装することを助言します。設計をできるだけ単純にしておけば後は勝手に面倒をみてくれるという考えなのです。RUP はそうした提案のリスクを評価することを勧めます。将来、システムまたはシステムの一部を書き換える必要が生じることがあっても、XP は可能性を今、計画するよりも適切で、多くの場合は安価であるとしています。システムによってはこの説は当てはまり、RUP を使用すれば、推敲フェーズにおけるリスクの検討は同じ結論に達します。RUP はこの事実をすべてのシステムについて保証するわけではなく、経験上、規模が大きく、複雑で、前例のないシステムについては破滅的になる可能性があると考えます。

まったく起こらないかもしれない将来の可能性についてあまりに注意し過ぎることは無駄になり得ることは確かですが、将来についてもある程度の注意を向けるのは賢明なことです。 継続してコードを書き換えたり、再構成したりできる会社がいったいどのくらいあるでしょうか。

どのプロジェクトでも、推敲の間に以下の 3 つのアクティビティーを行わなければなりません。

- **アーキテクチャーの定義、検証、ベースラインの確立を行う** — リスク・リストを使用して、方向づけフェーズの候補アーキテクチャーを開発します。私たちは開発予定のソフトウェアが実現可能であることを確信したいと考えます。選択したテクノロジーに目新しさがほとんどないか、またはシステムがあまり複雑でない場合、この作業にはそれほど時間はかかりません。 既存システムへの追加の場合、既存アーキテクチャーへの変更がまったくないならば、この作業は必要ないかもしれません。アーキテクチャー上の真のリスクが存在する場合、アーキテクチャーをそのままにしておきたくはありません。

  このアクティビティーの一部として、コンポーネントの選択と購買/構築/再利用に関する決定を行うことができます。多大な作業が必要になる場合は、別々のアクティビティーに分解することが可能です。

- **開発構想を洗練する** — 方向づけフェーズで開発構想書を作成しました。プロジェクトの実現可能性を見極め、利害関係者が時間をかけてシステムについてのレビューと見解を作成するうちに、開発構想書や要求に変更が生じることがあります。開発構想書や要求の改訂は一般に推敲の間に発生します。推敲の終わりまでに、アーキテクチャーと計画に関する決定を促す最も重要なユースケースを確実に理解するようになります。現在のアーキテクチャーに対して、現在の計画を実行してシステムを完成する場合、現在の開発構想が達成されることを利害関係者は同意する必要があります。後続の反復の間に変更量は少なくなりますが、要求管理の各反復に一定量の時間を割り当てるとよいでしょう。

- **作成フェーズ用の反復計画書の作成、ベースラインの確立を行う** — 現時点で、計画の詳細を記入します。各作成反復の終わりに、再度計画を検討し、必要があれば調整します。調整はたいていの場合、作業量が誤って見積もられたか、ビジネス環境が変化したか、要求が変更になったことが原因で発生します。ユースケース、シナリオ、技術的な作業に優先順位を付け、次にそれらを反復に割り当てます。各反復の終わりで利害関係者に価値を提供する作業中の製品が作成されるように計画します。

推敲の間にほかのアクティビティーを実行することができます。テスト環境を確立し、テストの開発を始めることをお勧めします。詳細なコードは存在しませんが、統合テストを設計し、場合によっては実装することが可能です。プログラマーは単体テストの開発準備ができており、プロジェクト用に選択されたテスト・ツールの使用方法を知っているでしょう。XP はコードの前にテストを作成するように推奨します。これは、特に、既存のコード本体に追加する場合はよい考えです。テストを実行するためにどのような選択をしたとしても、通常のテスト方式の確立は推敲において行います。

RUP により説明される推敲フェーズは XP の探索とコミット・フェーズの要素を含みます。斬新さや複雑さといった技術的なリスクの取り扱いに対する XP の手法は「スパイク」ソリューションです。 つまり、作業量を見積もるために一定の時間をかけて実験を行います。このテクニックは多くの場合効果的で、単一のユースケースまたはストーリーに、より大きなリスクが組み込まれていない場合、もう少し作業量を適用し、システムの成功と正確な作業量見積もりを確認する必要があります。

通常、方向づけフェーズの要求やリスク・リストといった成果物を推敲中に更新します。推敲中に現れる可能性がある成果物は以下のとおりです。

- **ソフトウェア・アーキテクチャー説明書 (SAD)** — SAD は、プロジェクトを通じて技術情報のただ 1 つのソースを提供する複合成果物です。推敲の終わりに、SAD は、アーキテクチャー上重要なユースケースと主要なメカニズム、設計要素の識別に関する詳細な記述を含むことがあります。プロジェクトが既存システムを拡張する場合、前の SAD を使用したり、この文書を使用しなくてもあまりリスクはないと判断したりすることが可能です。いずれの場合も、この文書につながる考慮されたプロセスを実行しなければなりません。

- **作成の反復計画書** — 推敲中に作成反復の数を計画します。各反復は割り当てられた特定のユースケース、シナリオ、そのほかの作業項目を含みます。この情報は反復計画内で把握され、ベースラインとして確立されます。推敲の終了基準の一部として計画をレビューして承認します。

非常に小規模で簡略なプロジェクトでは、推敲反復を方向づけと作成に結合することがあります。必須のアクティビティーは実行されますが、反復計画のリソースとレビューは減少します。

初期のユースケース・モデル

これは堅苦しく威圧的に思えるかもしれませんが、かなり単純明快です。ユースケースは、XP において顧客により作成される「ストーリー」に相当します。違いは、ユースケースが、目に見える価値を提供するシステム外のアクターや人、ものにより開始されるアクションの完全なセットであることです。ユースケースはいくつかの XP ストーリーを含む場合があります。プロジェクトの範囲を定義するために、RUP は方向づけの間にユースケースとアクターを特定することを推奨します。ユーザーの観点からアクションの完全なセットに焦点を当てることはシステムを価値ある部分に分割するのに役立ちます。これは、各反復 (初期の方向づけと推敲反復は除外される場合がある) の終わりに顧客に納品可能なものを提供できるように、適切な実装機能を識別するのに役立ちます。

RUP と XP は両方とも、システムの 80% が完了したということではなく、納入物として完了しているものは何もないということを保証してくれます。私たちは常に顧客に価値を提供するシステムをリリースしたいと考えています。

この時点のユースケース・モデルは、確証のある詳細をほとんど、またはまったく持たないユースケースとアクターを識別します。このユースケース・モデルは単純なテキストの場合もあり、手書きまたは描画ツールで描かれた UML (統一モデリング言語) ダイアグラムの場合もあります。このモデルにより、利害関係者に対する適切な機能が含まれていることや忘れている機能などがないことを確認し、システムの全体像を簡単に見ることができます。ユースケースは、リスク、顧客に対する重要性、技術的な難度といったいくつかの要因に基づき優先順位付けされます。

方向づけのすべての成果物は過度に格式ばったり、サイズを大きくしたりする必要はありません。正式にするか略式にするかは必要に応じて調整します。XP は計画とシステム受容に関するガイダンスを含みますが、RUP はプロジェクトの初期段階でそれらをもう少し追加します。この小さな追加は、さらに完全なリスクのセットに対応することにより大きな利益を生むことがあります。

作成

作成の目標はシステムの開発を完了することです。作成フェーズはある意味で、リソースの管理、コスト、スケジュール、品質を最適化するための業務管理に重点を置いた製造プロセスと考えられます。この意味で、方向づけと推敲におけるアイデアの開発から作成と移行における配置可能な製品の開発へと管理上の視点も切り替えます。

XP は作成に専念します。作成フェーズはコードを生成する段階です。XP のフェーズは計画の目的で存在しますが、XP の重点はコードの作成に重点を置きます。

各作成反復には以下の 3 つのアクティビティーがあります。

- **リソースを管理し、プロセスを制御する** — すべての人は誰が何をいつ行うかを知る必要があります。作業負荷が許容量を超えないこと、作業がスケジュールに従って進行していることを確認しなければなりません。
- **コンポーネントを開発し、テストする** — 反復のユースケース、シナリオ、そのほかの機能を満足させるのに必要なコンポーネントを構築します。単体テストと統合テストを通じてコンポーネントをテストします。
- **反復を評価する** — 反復の終了時に、反復の目標が満たされたかどうかを判断する必要があります。満たされなかった場合は、納品日に間に合うように範囲を再び優先順位付けし、管理する必要があります。

異なるシステムは異なるテクニックを必要とします。RUP は、ソフトウェア・エンジニアが適切なコンポーネントを作成するためのさまざまなガイドラインと支援を提供します。ユースケースと補足 (機能外) 要求の形態をとる要求はエンジニアが作業を行うのに十分な詳細さを持っています。RUP のいくつかのアクティビティーは、異なる種類のコンポーネントの設計、実装、テストに関するガイダンスを提供します。熟練したソフトウェア・エンジニアはこれらのアクティビティーを詳細に確認する必要はありません。経験の少ないエンジニアは最善の実践原則についての大きな支援を得ることになるでしょう。各

チーム・メンバーは必要に応じてプロセスへの関わりを調整することができます。ただし、すべての人はただ 1 つのプロセス知識ベースを使用します。

XP において、ストーリーは実装の原動力になります。「*Extreme Programming Installed*」(邦訳:「*XP エクストリーム・プログラミング導入編 - XP 実践の手引き*」) の中で Jeffries 氏達は、ストーリーはプログラマーとの「会話の取り決め」だと述べています。[1] 継続的で効果的な会話が望まれます。明確化を要する詳細は常に存在しますが、ストーリーにプログラマーが作業のほとんどを行うのに十分な詳細さが欠けている場合、それらの準備はできていないことになります。ユースケースはプログラマーが実装するのに足る詳細さを備えていなければなりません。多くの場合、プログラマーはユースケースの技術的詳細を書く手助けをしています。Jeffries 氏達は会話を記録し、ストーリーに添付するとも言っています。RUP はこの方法も提案しますが、必要に応じて略式化できるユースケース仕様という形態という点が異なります。会話の内容の把握と管理は必須の作業です。

XP の強みは作成にあります。ほとんどのチームに当てはまる非常に価値のある知識と指針があります。最も注目すべき XP の実践原則を以下に示します。

- **テスト** — プログラマーはコードの開発と共に継続的にテストを作成します。テストはストーリーを反映します。XP は最初にテストを作成するように強く勧めます。 なぜなら、これによりストーリーを深く理解し、必要な場合はさらに質問せざるを得なくなるからです。コードの前か後かは別にしても、必ずテストを作成することです。テストをテスト・セットに追加し、コードが変更されるたびに必ず実行します。
- **リファクタリング** — 動作を変更することなく継続的にシステムを再構成し、単純化を進めるか、柔軟性を追加します。これがチームにとって適切な実践原則かどうかを判断する必要があります。ある人にとっては単純でも、別の人には複雑になることがあります。あるプロジェクト・チームに非常に頭の切れるエンジニアが 2 人いて、互いのコードが複雑すぎると考えたことから、毎晩相手のコードを書き換えていたという例があります。副産物として、彼らは翌日になるとチームの残りのビルドも壊しつづけました。テストは有効ですが、チームがコーディング戦争に巻き込まれないほうが賢明です。
- **ペア・プログラミング** — XP はペア・プログラミングがより適切なコードをより少ない時間で生成すると主張します。これが事実である証拠があります[2]。この実践原則を実装する場合に考慮すべき人間性と環境上の要因が多数あります。プログラマーは喜んでこれを行うでしょうか。1 台のワークステーションで 2 人のプログラマーが効果的に作業できるような物理的な環境があるでしょうか。在宅勤務やほかの場所のプログラマーにはどのように対処しますか。
- **継続的統合** — 何度も、場合によっては 1 日に複数回、システムを統合し、ビルドします。これはコードの構造的な整合性を確かめるすぐれた方法であり、統合テストを通じて継続的な品質管理を可能にします。
- **共同所有** — 誰でもいつでもどのコードでも変更する権利を持っています。XP は、適切な単体テストのセットがこの実践原則のリスクを減らすという事実を信頼しています。誰もがすべてに親しむことの恩恵はある点を超えることができず、限界があるでしょう。 コードにして 1 万行または 2 万行の場合もあるでしょうが、5 万行未満であることは確かでしょう。
- **シンプル・デザイン** — リファクタリングと同様に、システムのデザインは継続的に修正され、複雑さが除去されます。繰り返しになりますが、機能しなくなる前にどこまで拡張する可能性があるのかを見極める必要があります。推敲中に時間をかけてアーキテクチャーを設計すれば、シンプル・デザインが実現され、より早く安定化します。
- **コーディング標準** — これは常に適切な実践原則です。標準の内容は問題でなく、標準を定め、すべての人が標準の使用に同意することが重要です。

RUP と XP は、反復を管理 (または進行) しなければならないという点で一致します。メトリックスは、チームにとっての最善を選択する助けとなるため、適切な計画情報を提供します。測定する項目には、期間、規模、障害の 3 つがあります。

---

[1] この説明は Allistair Cockburn によります。

[2] Strengthening the Case for Pair Programming, IEEE Software, July/August, 2000.

これにより関心のあるすべての種類の統計を得ることができます。XP は進捗を判断し、実績を予測するために使用する簡単なメトリックスを提供します。これらのメトリックスは、完了したストーリーの数、合格したテストの数、統計の傾向に集中しています。XP は最低限のメトリックスを使用することを強く主張します。測定を多くしたからといって必ずしもプロジェクト成功の可能性が高くなるわけではないためです。RUP は、測定の対象と方法に関するガイドラインを提供し、メトリックスの例を用意しています。どの場合でも、メトリックスはシンプルで客観的、収集が容易、解釈が容易、かつ誤って解釈されにくいものである必要があります。

作成反復で出力される成果物は何でしょうか。反復が初期の作成反復か後期の作成反復かによって、以下のいずれかを作成します。

- **コンポーネント** — コンポーネントとは、ソフトウェアのコード (ソース、バイナリー、または実行可能ファイル)、あるいは情報を格納したファイル (起動ファイルや ReadMe ファイルなど) を指します。また、コンポーネントは、複数の実行可能ファイルから成るアプリケーションなど、ほかのコンポーネントの集合体である場合もあります。
- **トレーニング教材** — システムのユーザー・インターフェース側面が強い場合、ユースケースに基づき、ユーザー・マニュアルやトレーニング教材の予備原稿をなるべく早い時期に作成します。
- **導入計画書** — 顧客はシステムを必要としています。導入計画は、製品のインストール、テスト、ユーザー環境への効果的な移行に必要な一連の作業を説明します。Web 中心のシステムの場合、導入計画の重要性が高いことがわかっています。
- **移行フェーズ反復計画** — ソフトウェアをユーザーに配置する時期が近づくと、移行フェーズ反復計画を完了し、レビューします。

本当にコードだけの問題か

RUP と XP の違いはアーキテクチャーに対する手法の違いだけではありません。その 1 つは設計を伝達する方法です。XP はコードが設計であり、設計がコードであるとしています。コードが常にコードそのものに一致していることは事実です。私たちはコード以外に、設計を把握し、伝達する作業にも時間をかける価値があると考えます。例としてちょっとした話を紹介しましょう。

あるエンジニアが、設計がコードに含まれ、コードが設計情報を確認できるただ 1 つの場所であるソフトウェア・プロジェクトを 2 つ経験しました。これらのプロジェクトはいずれもコンパイラーが関係していました。1 つは Ada コンパイラーのオプティマイザーを向上させ、管理するプロジェクトで、もう 1 つはコンパイラーのフロントエンドを新規のプラットフォームにポーティングし、サード・パーティーのコード・ジェネレーターをリンクするプロジェクトでした。

コンパイラーのテクノロジーは複雑ですが、よく知られています。両方のプロジェクトで、エンジニアはコンパイラー (またはオプティマイザー) の設計と実装の概要を知りたいと思いました。それぞれの場合で、エンジニアは厚さが数 cm にもなるソース・コード・リストの山を渡され、「これを見てくれ」と言われました。エンジニアはてっきりきちんとしたダイアグラムに説明文が添えられた資料をもらえるものと思っていました。オプティマイザーのプロジェクトは結局完了しませんでした。コンパイラーのプロジェクトは無事完了し、品質の高いコードが生成されました。 これは、コードと共に拡張されたテスト・セットが開発されたためです。エンジニアは何日もかけてデバッガーでコードを追い、その仕様を理解しようとしました。わずかな疲労という個人的なコストやチームに対するコストはそれに匹敵するものではありませんでした。XP が提案する最高労働時間 40 時間という選択肢はなく、作業を完了させるために思い切った労力をつぎ込みました。

コードしか持たないということの第一の問題は、たとえ十分な説明が付いていても、コードから解決する問題を知ることはできず、わかるのは問題への解決策だけだということです。要求の文書の中には、当初のユーザーや開発者がいなくなった後でも、元の目標を説明する際にたいへん役に立つものがあります。システムを保守するには、元のプロジェクト・チームがどんなことを思い描いていたかを知る必要がよくあります。 いくつかの高レベルの設計書も同様です。多くの場合、コードは抽象化できるレベルとしては低すぎて、システムが全体として何をしようとしているのかを理解するのが難しいものです。 このことは、オブジェクト指向システムに特に当てはまり、クラスを単に見るだけでは実行のスレッドを追うことは困難または不可能です。設計文書は、後で問題が起こった場合に参照すべき場所を案内してくれます。 そして、問題はいつも後で起こるものなのです。

この話の教訓は、設計書の把握と管理に費やす時間が本当に役に立つということです。誤解のリスクは減り、開発のスピードは上がります。XP の手法は設計のスケッチを数分で終えるか、CRC カードを使用することです[1]。チームはこれらを保守せず、コードの作業に取り掛かります。ここには、作業が十分単純で、既に進め方がわかっているという暗黙の前提が存在します。現在、進め方を理解していても、次にやってくる人がそれほど幸運だとは限りません。RUP はこれらの設計成果物の把握と保守にもう少し時間をかけることを推奨します。

移行

移行フェーズの焦点は、エンド・ユーザーがソフトウェアを入手し、使用可能であることの確認です。移行フェーズは、リリースの準備として製品をテストし、ユーザーのフィードバックに基づいて微調整します。ライフ・サイクルのこの時点におけるユーザーのフィードバックは、製品の微調整、構成、インストール、使いやすさの問題に集中する必要があります。

初期リリースを早め、リリースの回数を増やすことはよい考えです。しかし、リリースとは何でしょうか。XP ではリリースの定義があいまいで、商用ソフトウェアのリリースに必要な製造の問題を説明しません。社内使用製品の場合はいくつかの問題処理を省略できるかもしれません。ただし、その場合でも、マニュアルやトレーニングは必要です。サポートや変更管理についてはどうでしょうか。現場の顧客がこれも管理してくれると期待するのは果たして現実的でしょうか。Bruce Conrad 氏は XP に関する InfoWorld の論評[2]の中で、顧客は変更され続けるソフトウェアを欲しがらないと指摘しています。顧客に対して変更を迅速に適用することのメリットと、変更やそれに伴う不安定さが持つデメリットをはかりにかけなければなりません。

リリースすることを決定したら、エンド・ユーザーにコード以外のものも提供する必要があります。移行におけるアクティビティーと成果物は、ソフトウェア開発プロセスのこの段階の案内役になります。アクティビティーは使用可能な製品を顧客に提供することに重点を置きます。重要な移行アクティビティーは以下のとおりです。

- **エンド・ユーザー・サポート文書を完成させる** — このアクティビティーはリストの項目をチェックするくらいの簡単なものになる場合がありますが、顧客に対するサポートの準備が組織内に整っていることを確認する必要があります。
- **顧客環境で納入する製品をテストする** — 自社で顧客環境をシミュレートできる場合は実行します。できない場合は顧客に赴き、ソフトウェアをインストールして動作を確認します。顧客に「しかし、弊社のシステムでは動作します」と言うようなつまらない状況は作りたくありません。
- **顧客のフィードバックに基づき製品を微調整する** — 可能であれば、1 回以上のベータテスト期間を設け、限られた数の顧客にソフトウェアを納品します。これを行う場合、ベータテスト期間を管理し、顧客のフィードバックを「最終段階」で検討する必要があります。
- **エンド・ユーザーに最終製品を納品する** — ソフトウェア製品のタイプとリリースによって、対応しなければならないパッケージ、製造、そのほかの生産問題はさまざまです。ソフトウェアをディレクトリーに放り込み、そこに用意ができた旨のメールを顧客に送りつけるだけで済むことはめったにありません。

ほかのほとんどのフェーズと同様に、プロセスの正式さや複雑さはさまざまです。しかし、顧客への配置の詳細に注意を払わなければ、何週間から何カ月にも渡るすぐれた開発作業の苦労も水の泡となり、製品を、目標とする市場の失敗作にしてしまう可能性があります。

移行フェーズではいくつかの成果物を生成する場合があります。製品に将来のリリース予定がある場合は、次のリリースに向けた機能と欠陥修正の特定を開始します。任意のプロジェクトについて、重要な成果物は以下のとおりです。

---

[1] CRC (クラス、責任、協力) カードは、Kent Beck 氏と Ward Cunningham 氏により開発されたもので、実践者にオブジェクト指向設計を教えます。

[2] http://www.infoworld.com/articles/mt/xml/00/07/24/000724mtextreme.xml

- **導入計画書** — 作成フェーズで開始した導入計画書を完成させ、顧客納品のためのロードマップとして使用します。
- **リリース・ノート** — エンド・ユーザー向けの最終段階の指示を持たないソフトウェア製品はまれです。リリース・ノートを計画し、読みやすく、一貫性のあるテキストを準備します。
- **トレーニング教材とマニュアル** — これらの資料はさまざまな形態をとります。すべてオンラインで提供しますか。チュートリアルを用意しますか。製品ヘルプは完全で使いやすいですか。あなたが知っていることを顧客が知っているとは限りません。プロジェクトの成功は顧客の成功にかかっています。

まとめ

ソフトウェアの構築は単にコードを書くことにとどまりません。ソフトウェア開発プロセスは、顧客に品質を提供するために必要なすべてのアクティビティーに重点を置きます。完全なプロセスは必ずしも大掛かりではありません。このホワイト・ペーパーでは、プロジェクトに必須のアクティビティーと成果物に焦点を当てることにより、小規模でも完全なプロセスを実現する方法を述べてきました。プロジェクト上のリスクを軽減するのに有効なアクティビティーを実行し、成果物を生成します。プロジェクト・チームと組織の必要に応じて、プロセスと正式さを調整します。

RUP と XP は必ずしも排他的ではありません。両者の手法からテクニックを取り込むことにより、今よりも早く、より高品質のソフトウェアを提供するのに役立つプロセスを確立できます。Robert Martin 氏は RUP に準拠する dX プロセス というプロセスを説明しています。[fn] これは、RUP フレームワークから構築されたプロセスのインスタンスです。

すぐれたソフトウェア・プロセスは業界で実証された最善の実践原則を取り入れます。最善の実践原則は現実のソフトウェア開発組織で長期間テストを経てきたものです。XP は今日多くの注目を集めている手法です。XP はコード中心であり、最小限のプロセス・オーバーヘッドと最大限の生産性を提供します。XP には、適切な状況における検討と採用を保証するテクニックが多数あります。

XP はストーリー、テスト、コードに重点を置きます。 つまり、XP は計画についてある程度説明しますが、計画の把握は軽く扱います。XP では、「いくつかのカードを使用して CRC 設計を行うか、UML をスケッチする」とか、「使用されない文書やそのほかの成果物は作成しない」というように、ほかの資料を作成することもできますが、あくまでも一時的なものとして扱います。RUP では、開発計画を策定し、更新する際、有用で必要なものだけを作成することを推奨し、それらの実体を特定します。

RUP は完全なソフトウェア開発ライフ・サイクルに対応するプロセスです。RUP は何千というプロジェクトで進化してきた最善の実践原則に焦点を当てます。最善の実践原則につながる新しいテクニックの調査と創案を奨励します。最善の実践原則が生まれ、RUP に取り入れることを期待しています。

付録 A:Rational Unified Process

Rational Unified Process または RUP はソフトウェア開発に対する規則正しい手法を提供します。RUP は Rational により開発され保守されるプロセス製品です。RUP には異なる種類のソフトウェア・プロジェクト向けの標準ロードマップが付属します。RUP は、ソフトウェア開発用のほかの Rational ツールを使用する際に役立つ情報も提供しますが、組織に効果的なアプリケーションを実現するために Rational ツールが必須というわけではなく、ほかのベンダーの製品を統合することも可能です。

RUP はソフトウェア・プロジェクトのすべての側面に関する指針を提供します。RUP は特定のアクティビティーの実行や特定の成果物の生成を強制しません。RUP が提供するのは、実際の組織に適用可能なものを判断するための情報とガイドラインです。また、RUP は、標準のロードマップがプロジェクトまたは組織に適合しない場合に、プロセスをカスタマイズするのに役立つガイドラインも提供します。

RUP は、新しいソフトウェアの開発に特有のリスクを軽減する方法として、現代のソフトウェア開発における最善の実践原則を採用することに重点を置きます。こうした最善の実践原則は以下のとおりです。

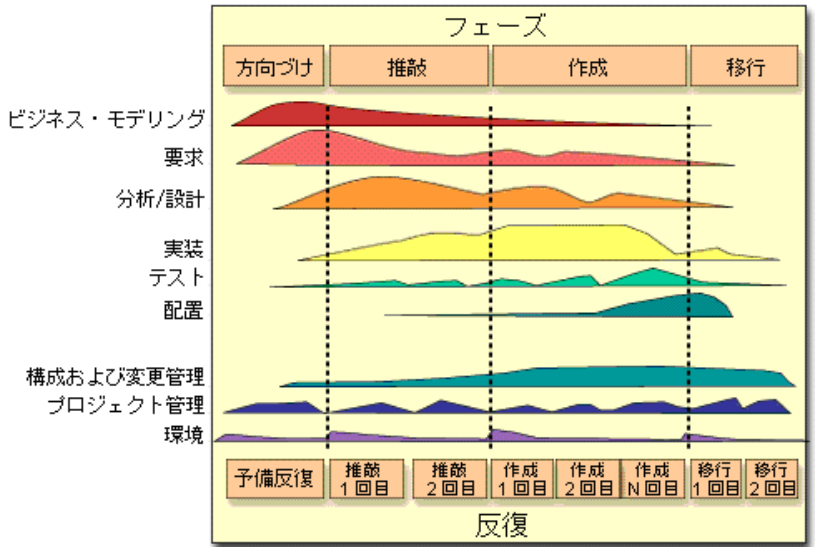
1. 反復的に開発する

2. 要求を管理する
3. コンポーネントベース・アーキテクチャーを使用する
4. ビジュアルにモデリングする
5. 継続的に品質を検証する
6. 変更を管理する

これらの最善の実践原則は以下の Rational Unified Process 定義に組み込まれています。

- ロール — 実行されるタスクと所有される成果物のセット
- 作業分野 — 要求、分析と設計、実装、テストといったソフトウェア開発作業の重点分野
- タスク — 成果物が生成され、評価される方法の定義
- 成果物 — タスクの実行中に使用、生成、または変更されるワーク・プロダクト

RUP は任意のソフトウェア開発プロジェクトの 4 つのフェーズを特定する反復的プロセスです。時間の経過と共に、プロジェクトは方向づけ、推敲、作成、移行の 4 つのフェーズを通ります。各フェーズには 1 回以上の反復が含まれ、実行可能形式を生成しますが、システムとしては不完全な場合があります (方向づけフェーズは除外される場合がある)。各反復中に、いくつかの作業分野のアクティビティーをさまざまな詳細レベルで実行します。次の図は RUP の概要を示しています。



RUP 概要図

「*The Rational Unified Process, An Introduction, Second Edition*」(邦訳:「*ラショナル統一プロセス入門 第 2 版*」) は RUP の概要をわかりやすく述べています。RUP の詳細と評価については、Rational Web サイト: www.rational.co.jp を参照してください。

付録 B:エクストリーム・プログラミング

エクストリーム・プログラミング (XP) は、1996 年、Kent Beck 氏により開発されたソフトウェア開発規律です。XP は、コミュニケーション、シンプルさ、フィードバック、勇気という 4 つの価値に基づいています。XP は、開発を進める間にオンサイトに顧客を招くことにより、顧客と開発チーム・メンバー間の継続的なコミュニケーションを重視します。オンサイトの顧客は構築の対象と構築の順序を決定します。シンプルさは、継続的にコードを再構成し、最低限の非コード成果物のセットを作成することにより実現します。頻繁なリリースと継続的な単体テストはフィードバックのメカニズムです。勇気とは、一般的でないことでも正しいことを行うという意味です。 また、できることとできないことについて正直になることでもあります。

これらの 4 つの価値をサポートする 12 個の XP 実践原則を以下に示します。

- **計画ゲーム** — 優先順位付けされたストーリーと技術的な見積もりにより、次回リリースの機能を決定します。
- **小規模リリース** — 規模の小さな増分バージョンとしてソフトウェアを何度も顧客にリリースします。
- **比喩** — 比喩はシステムが動作するしくみを示す簡単な共有ストーリーまたは説明です。
- **シンプル・デザイン** — コードをシンプルに保つことにより、設計をシンプルに保ちます。コード内の複雑さを継続的に探し、見つけたらすぐに取り除きます。
- **テスト** — 顧客はストーリーをテストするためにテストを作成します。プログラマーは、コード内で中断する可能性があるすべてのものをテストするためにテストを作成します。コードを作成する前にテストを作成します。
- **リファクタリング** — これはコードから重複と複雑さを取り除く簡略化テクニックです。
- **ペア・プログラミング** — プログラマーは 2 人 1 組になり、1 台のコンピューターを使用してすべてのコードを開発します。1 人がコードを作成、または実行すると同時に、もう 1 人が正しさとわかりやすさについてコードをレビューします。
- **共同所有** — すべての人がすべてのコードを所有します。これはだれもがすべてのコードをいつでも変更できることを意味します。
- **継続的統合** — 任意の実装作業が完了するたびに、1 日に何度もシステムをビルドして統合します。
- **週 40 時間** — プログラマーは疲労していると最高の効率で働くことができません。2 週連続で残業することは許されません。
- **オンサイト顧客** — 実際の顧客が開発環境でフルタイムで働き、システムの定義、テストの作成、質問の回答を支援します。
- **コーディング標準** — プログラマーは統一されたコーディング標準を採用します。

XP を説明した書籍として、以下の 3 冊が現在入手可能です。

1. 「eXtreme Programming Explained」(邦訳:「XP エクストリーム・プログラミング入門」、永田 渉、飯塚 麻理香 訳、長瀬 嘉秀 監訳、ピアソン・エデュケーション)
2. 「Extreme Programming Installed」(邦訳:「XP エクストリーム・プログラミング導入編 - XP 実践の手引き」、平鍋健児 、高嶋優子、 藤本聖 訳、ピアソン・エデュケーション)
3. 「Planning Extreme Programming」(邦訳:「XP エクストリーム・プログラミング実行計画」、飯塚 麻理香 訳、長瀬 嘉秀 監訳、ピアソン・エデュケーション)

XP の詳細を紹介するいくつかの Web サイトもあります。

Rational®
the software development company

Dual Headquarters:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Toll-free: (800) 728-1212
E-mail:info@rational.com
Web:www.rational.com
International Locations:www.rational.com/worldwide