

XP(Extreme Programming): ソフトウェア開発プロセスの新潮流 ～ 前編 : XP 概要とその周辺 ~

初出 : 情報処理 Vol.43, No.3 Mar.2002
(株) 永和システムマネジメント
平鍋健児(hiranabe@esm.co.jp)

利用上の注意事項 : ここに掲載した著作物の利用に関する注意 本著作物の著作権は(社)情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。

1999 年に刊行された "Extreme Programming Explained: Embrace Change"[Beck99](邦訳『XP エクストリーム・プログラミング入門』)以降, XP は軽量かつ拙速なソフトウェア開発手法として, プログラマはもとよりプロジェクト管理者, 品質管理者, 顧客, そして経営者からも大きな注目を集めるにいたっている。

最初, 私にとって XP のテーマは「プログラマの復権」あるいは「重量級方法論へのアンチテーゼ」に見えた。しかし, Kent Beck 自身と出会い, XP の講演やメーリングリストで多くの人と話す機会を得たことで, XP の人気の秘密は上記以外にもあることを認識するに至った。XP のテーマは見る人によって実に多様である。それらは, 「低リスクの開発手法」, 「顧客と開発の双方にお得な契約方法」, 「やりがいのある職場の作り方」, 「高品質を保つ開発手法」, 「開発チームからベストを引き出す顧客の心得」などなど, さまざまなのだ。

本記事「前編 : XP 概要とその周辺」では, XP の概要とその基本的な考え方を解説して, その魅力の一端を明らかにしたい。また, その歴史や周辺にも触れてみたい。

一方で XP は, その過激さから, ともしればカルトとして見られてしまいがちな側面もある。しかし, もう一方ではきわめて現実的な開発手法として, 日本でも実例を増やしながらかつ着実に有効性が検証されてきていると言える。次号の記事「後編 : XP 実践事例の紹介」では, 実際の実践レポートを掲載する予定である。

従来の開発と XP

まず, 従来型開発のイメージを絵にしてみた(図1)。記事は XP を中心にしているため, たぶん従来型に対して誇張した批判的な書き方になっていることはご容赦願いたい。

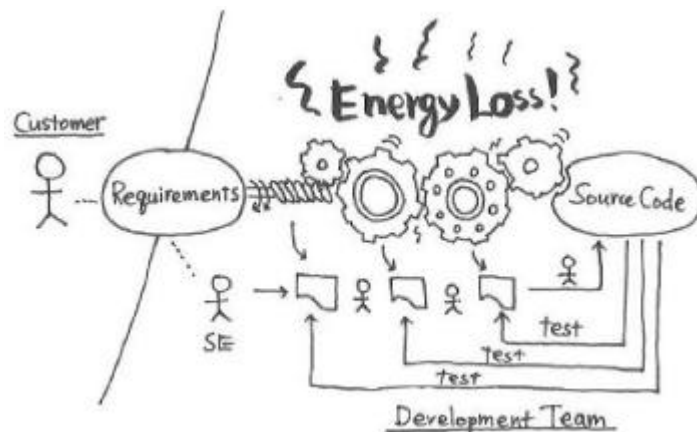


図 1 従来型開発

従来型開発においては、「SE」(エス・イーと読むらしい)と呼ばれる役割のエンジニアが顧客と話し合っ要求を顧客から引き出し、仕様を記述する。そして、その要求は工程に基づいてブレークダウンされ、最終的なコードに落ちて行くことになる。この過程には、歯車が幾つもあり大量のドキュメントが生成される。典型的には、基本設計書、機能設計書、詳細設計書、テスト仕様書などと呼ばれる一群のドキュメントが中間成果物となり、これらの作成活動やレビュー活動が中間生産物を生み出す歯車となる。いわゆる、「重量級プロセス」だ。

仕様が非常に安定した大規模開発であれば、この手法はリスクが低く着実な方法と言えるかもしれない。しかし、要求が安定しておらず、顧客自身が仕様を「学んでいる」過程にあったり、ビジネスの動きが速く仕様が流動的であったりする場合には、うまくいかないだろう。むしろ、現代の機敏なビジネスに対応しようとする開発では、後者のような場合が支配的なのではないだろうか。

典型的なパターンとして、リリースを行った後になって、顧客と開発側はプログラムの修正が「仕様変更かバグか」をめぐって敵対関係になってしまう。また、最終的なコードが元の設計からずれてしまっていて、中間成果物のメンテナンスが不能になる。プログラマが実装中によりよい設計を見出しても、それを上流工程の設計にフィードバックすることは困難であり、おかしな設計のままコーディングしてしまう。テスト工程はさらに悲惨であり、動いているコードを修正することはご法度である。修正はその場しのぎでコードにパッチをあてるように行われ、つぎはぎだらけのプログラムが完成する。このようにして開発されたプログラムを、実際の現場で保守していくことの困難さは想像に難くないだろう。

こういった開発手法の問題点は、ドキュメント中心でコミュニケーションを排除する傾向にあること、上流の工程に対して実装やテスト段階で見つかった問題点がフィードバックしにくいこと、組織の壁に阻まれて、開発者個人の判断で重要な情報にアクセスできないこと（顧客に質問をするには、Q & A 票を書かねばならないなど）がある。この開発モデルには「エネルギーロス」が多すぎるのだ。

これに対して XP の開発イメージは、図 2 のようになる。

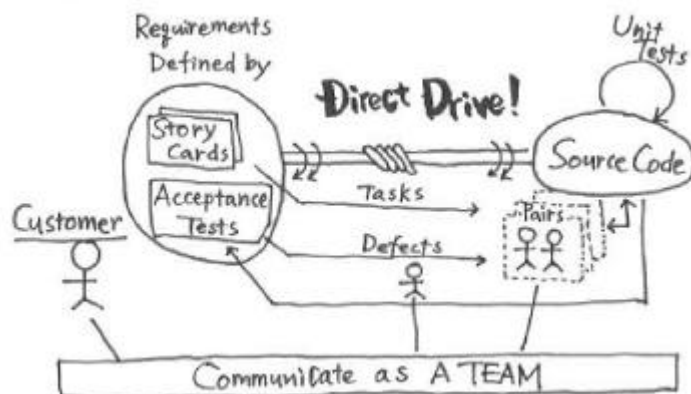


図 2 XP 型開発

このモデルでは、要求がソースコードを「ダイレクトドライブ」で駆動する。顧客は「ストーリーカード」と呼ばれる仕様を簡潔に記述したカードと、「受け入れテスト」と呼ばれるテストで仕様を定義する。顧客自身が「このテストを通れば仕様を満たしている」というテストを通じて要求を定義するのだ。ストーリーは「タスク」にブレイクダウンされ、プログラマ・ペアによって実装される。すべての製品コードはペアプログラミングによって書かれる。しかも、ここでのコーディング手法は「テストファースト」だ。まずクラスの仕様を表現するユニットテストを書き、それを通す製品クラスを書くことによって開発を進める。

XP では、顧客はストーリーの開発順序や仕様そのものを途中で変更することが許されている。もちろん開発者の見積りを尊重し、同じ費用のストーリーと入れ替えるのだが、動くコードとしてフィードバックが即座に行われることで、顧客が仕様を「学んで」いく過程がプロセスに組み込まれているのだ。また、プログラマーは小さなユニットテストを書き、それをパスさせていくことで小さな達成感を得ながら一步一步前に進む。プログラマーもまた、開発をしながらストーリーの難易度を学び、それを見積りに反映していく。そして何より、顧客と開発者がコミュニケーションを基礎とする「チーム全体(whole team)」を形成

しているのが、XP の最大の特徴だ。

要求を「変化の湧き出し口(source)」、製品プログラムを「変化の吸い込み口(sink)」と考えたときに、XP は要求の変化を source から sink へと最速で伝播させるモデルと捉えることもできる。その変化を伝える「媒体(media)」は、人間のコミュニケーションであり、これが、「変化ヲ抱擁スル(Embrace Change)」XP の開発モデルである。

4 つの価値

XP では、チームが何に重きを置くかの判断基準として、以下の「4 つの価値」を示している。

◆ コミュニケーション

コミュニケーションは 4 つの価値の中で最も重要である。XP は顧客と開発者、また開発者同士のコミュニケーションがあって初めて成り立つ。

◆ シンプルさ

XP の基本設計戦略は「シンプルさ」である。今必要とされるストーリーのみを視野に入れて設計する。将来を考えて複雑さを持ち込んではいけない。将来のことは、それが本当に必要になったときに初めて考える。この根底には、変更のコストは時間と共に増加していくものではなく、ある一定のレベルで抑えることが可能であるという XP の仮説がある(図3)。この仮説は「XP を行うことで得られ」また、「XP を行うために必要である」という相補的関係にあることに注意したい。

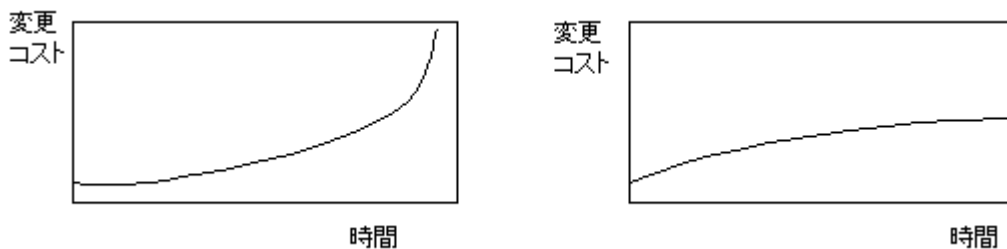


図 3 時間と変更のコスト

◆ フィードバック

XP ではフィードバックを常に求める。テストを中心とした開発を行うことにより、システムからのフィードバックを得る。また、リリースを頻繁に行うことにより顧客からのフィードバックを得る。フィードバックは、開発者の不安を取り除く。

◆ 勇気

コミュニケーション，シンプルさ，フィードバックの 3 つの価値は，開発者に勇気を与える．実装済みの多くの機能に影響を与えるような変更を行ったり，うまくいきそうになりアプローチのコードを捨てたりすることを，勇気を持って行うことができる．

XP のプラクティス

XP ではプロジェクトが従うべき実践項目を 14 個¹のプラクティスとして定めている．以下に，それぞれを簡単に紹介しよう．

「プラクティス」とは，経験に基づきその有用性が立証されてきた実践項目を指す．XP では 14 個のプラクティス全てを最大限に実践することを基本とし，ゆえにエクストリーム (= 極限) という名前がある．プラクティス全てを最大限に実践することにより，個々のプラクティスが相互に補強しあい，プロジェクトを安定させ，かつ，生産性を高くする．

◆ オンサイト顧客(On-Site Customer)

XP は「顧客」を含めたチーム全体でプロジェクトを進める．顧客がチームに入り，開発者のすぐそば(オンサイト)で一緒に仕事をする．ここが他の開発方法と大きく異なる所だ．顧客はビジネスの代理人であり，最大のビジネス価値が得られるようにチームに対する要求を定義し，そのプライオリティを設定する．また，プログラマからの質問にその場で答える．

◆ 計画ゲーム(Planning Game)

XP の開発は，顧客とプログラマが共同でシンプルな計画を立てることから始める．XP では，開発計画を「リリース」とよぶ 2~3 ヶ月単位の期間で構成する．それぞれのリリースは，さらに「イテレーション」とよぶ 2~3 週間単位の期間で構成する．開発計画は，顧客が要求をストーリーとして提示することから始まり，そのストーリーを元にリリース計画とイテレーション計画を立てる．顧客はストーリーを選ぶ権利，優先順位をつける権利があり，開発者はそれを見積もる権利がある．この計画は開発の間，継続的に行われる．

ストーリーが大きく見積りが 3 週間を超えてしまう場合，顧客にストーリーの分割を依頼

¹ XP のプラクティスとしては従来 12 個のものが知られている[Beck99]が，ここでは，その後に Kent Beck 自身によって追加された 2 つを加えて 14 個としたものを紹介する．なお，その後 Ron Jeffries が論文 "What is Extreme Programming?" [Jeffries01] でさらに 13 個に整理しなおしている．

することができる。これにはストーリーの粒度をある程度統制できる利点がある。また、開発側の技術的経験が不足して見積もれない場合、「スパイク」と呼ばれる大まかな解決を試してみることがある。スパイクの中でも、最初のアーキテクチャの「種」となるものを「アーキテクチャスパイク」と呼び、必要であればリリース計画に先だって実験が行われる。

◆ テスティング(Testing)

XP はテストを重視する。テストには、プログラマが行う「ユニットテスト」と顧客が受け入れの仕様を明確にする「受け入れテスト」がある。XP の顧客は、提示した機能が正しく実装されていることを確認するために「受け入れテスト」を定義する。この受け入れテストは必ず自動化され、システムのリリース時に常に実行される。チームはこの受け入れテストにより、必要とされる機能が正しく実装できたこと、および、既に実装済みの機能に悪い影響を与えていないことを確認する。

実際には、顧客にテストの作成を依頼することが困難な場合もある。その場合でも、ストーリーから「テストシナリオ」を作成することでテストを定義してもらう。

プログラマはコーディングに先立ってクラスの「ユニットテスト」を書く。そして、そのユニットテストにパスするまでコーディングとテストを繰り返す。このユニットテストはコードと一緒にリリースされ、システムにコードを追加するときは、そのシステムのこれまでに書かれた全ユニットテストをパスする必要がある。また、プログラムを作成する前にテストを書くことにより、その機能を「使う」ことに焦点を当てた良い設計へ繋がる。この新しい開発スタイルは、「テストファースト」開発と呼ばれる。

◆ 小さなリリース(Small Releases)

XP では 2~3 ヶ月のリリースごと、そして 2~3 週間のイテレーションごとに実際に動くプログラムを提供する。小さな単位で頻繁に顧客に提供し、実際に使ってもらうことにより、顧客に対して最短の時間でビジネス価値を提供し、そして顧客から最大のフィードバックを得る。

◆ シンプルな設計(Simple Design)

XP ではシステムを可能な限りシンプルに設計する。求められている機能を実装するために最もシンプルな設計を素早く行い、そしてテストと実装を進める。XP には'YAGNI'という標語がある。「You Are not Going to Need It」(多分それは必要にならない)の略であり、実装するのは最低限必要な機能のみという掟である。必要と「なるであろう」機能については実装を行わない。何故ならばそれは現時点で不要な機能であり、コード量を増やして

可読性を落とし、メンテナンスを難しくしてはいけない。そのコードが必要かどうかは現時点で可能性は高くても不明であり、不明なことに前払いで投資しないのが XP 流である。

設計に迷ったら、「CRC カード」を用いた短時間の設計ミーティングをその場で行う。このカード 1 枚を 1 つのクラスに見立て、設計者間でロールプレイングゲームを行うことでクラスの責務を明らかにしていく活動だ。

◆ ペアプログラミング(Pair Programming)

XP でのコーディングは常に 2 人がペアになって 1 台のコンピュータに向かって行う。ペアでプログラミングを行うことにより、コードは常にレビューされ、単純ミスが減り品質が上がる。

ペアプログラミングの作業コストは 1 人(シングル)で行う場合と比較して 200%になると考えてしまいがちだが、論文 *"The Costs and Benefits of Pair Programming"* [Cockburn99] の実験によると、そのコストは 115%であるとの結果が出ている。しかもシングルと比較して、ペアプログラミングのコードの欠陥率は 15%低く、また、コードサイズは 80%(おそらくは良い設計がなされている)との結果も出ており、作業時の 15%のオーバヘッドは、これらによって相殺されると考えられている。

また、ペアでプログラミングを行うことによりチーム内での知識の共有や各自のスキルアップが自然に行われる。このことはプロジェクトのトラックナンバー²を増やす上でも重要といえる。

◆ リファクタリング(Refactoring)

XP のシンプルな設計はコードの再設計手法である「リファクタリング」 [Fowler+99] によって洗練されていく。不用意に導入された複雑さは、リファクタリングによって取り除かれる。ここで、ユニットテストと受け入れテストは、リファクタリングで間違った変更をしたのではないかという不安からチームを解放する。

" In XP, the software life-cycle is: Analysis, Test, Code, Design. "

-- **Ralph Johnson**

この表現は必ずしも正確ではないにしろ、テストファースト開発とリファクタリングに

²トラックナンバー：トラックに轢かれるとプロジェクトが継続できなくなるメンバーの人数。最悪は 1 で、これは 1 人のスーパーマンがプロジェクトのキーとなっているケース。

よる進化型設計という XP の特徴をうまく表現している。

◆ コードの共同所有(Collective Code Ownership)

XP ではコードの所有者を決めない。全てのコードは全員で共有し、誰でもコードを自由に変更することができる。これにより、担当者が忙しくて仕事が進まない、とか、本来担当者のコードに入れるべき機能を自分のコードでカバーしてしまう、という不健全な設計がなくなる。また、多数の人の目によってコードがチェック(レビュー)されるため、品質が向上する。

◆ コーディング標準(Coding Standard)

XP では全てのコードフォーマットはプロジェクトのコーディング標準にのっとる。これで、ペアプログラミング時にインデントなどの本質的でない議論がなくなり、コードの共同所有を円滑にする。できあがったコードは、一人の優秀なプログラマが書き上げたような見栄えになる。

◆ 継続的インテグレーション(Continuous Integration)

XP では 1 日に何回でもインテグレーション(結合)を行う。ユニットテストが完了したタスクは、すぐにシステムに結合され、全ユニットテストが行われてリポジトリに登録される。頻繁に結合を行うことにより、プログラマ間の変更の衝突が起きにくくなり、誤って他の機能を壊してしまっても、すぐに検出することができる。

◆ メタファ(Metaphor)

XP ではシステムの動きを理解するために、チーム全員がメタファ(比喩)と呼ばれるビジョンを共有する。メタファはシステムがどのように動くかを説明した、誰にでも分かるシンプルな例え話だ。ある意味、アーキテクチャのようなグローバルな構造を創発させる役目を果たす。チームはこのメタファを使って互いのコミュニケーションをよりスムーズにできるし、クラスの命名などにも一貫性を持たせることができる。

◆ 週 40 時間労働(40-Hour Week)

XP ではいつまでも持続可能なペースで作業を行う。残業続きの開発では、決してよい結果は生まれない。

◆ オープンワークスペース(Open Workspace)

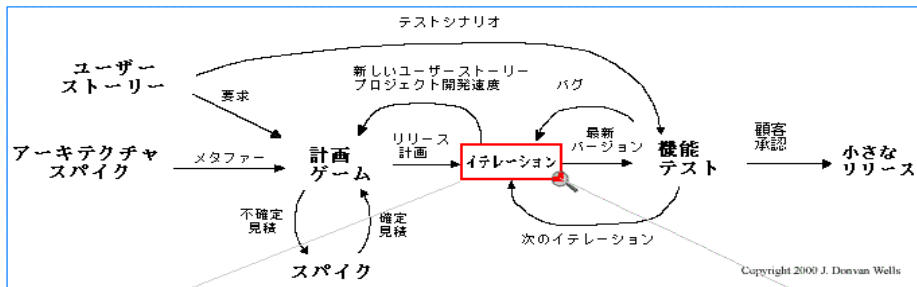
XP はコミュニケーションを促進するために、オープンな作業環境を必要とする。その場ですぐに設計のディスカッションが行える場だ。ホワイトボードは必須のコミュニケーションツールである。

◆ 日ごとのスキーマ移行(Daily Schema Migration)

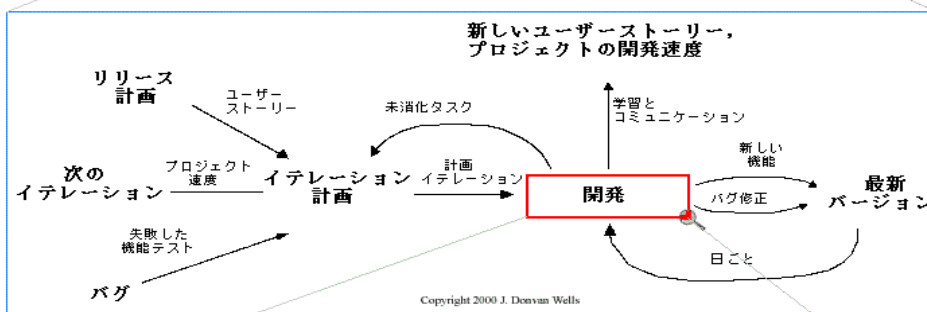
XP では永続性データ（データベースなど）を扱う場合，そのスキーマは日ごとに最新のものを使う．後で変更をまとめて入れよう，などという先延ばしは禁物だ．

これら 14 のプラクティスは，XP プロセスを生成する「パターン言語」と捉えることもできる．XP プロセスとしては，図 4 のような形が典型的[Wells99]だ．

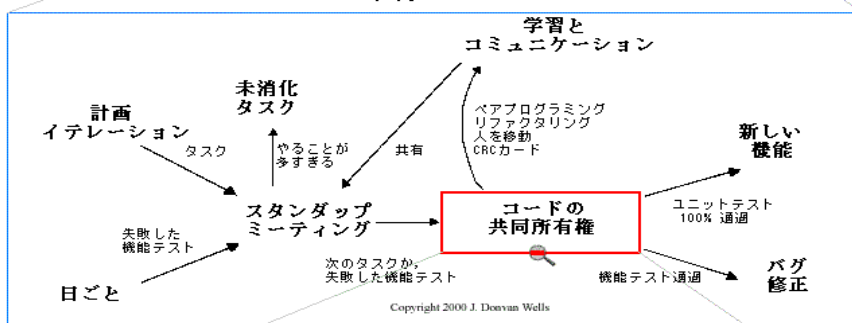
XPプロジェクト



イテレーション



開発



コードの共同所有権

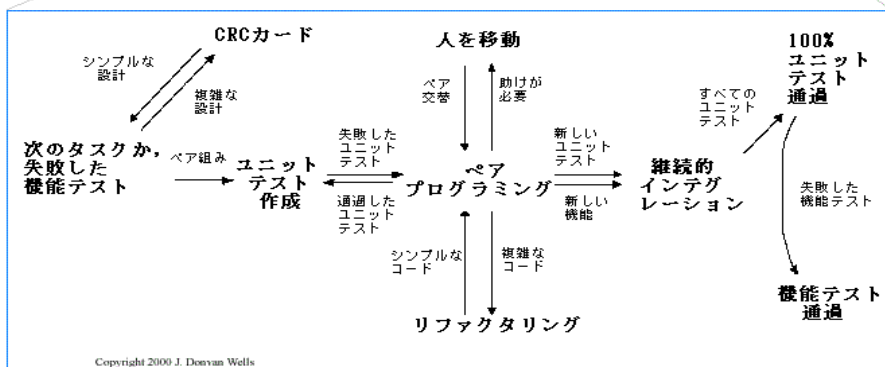


図 4 XP プロセス例

XP の背景

では、簡単に XP の歴史的背景に触れてみよう。

XP は Smalltalk のコミュニティから生まれている。生みの親は Kent Beck と Ward Cunningham。そして、1996 年に伝説的なプロジェクト C3(Chrysler Comprehensive Compensation)において、Kent Beck は Ron Jeffries, Martin Fowler らと共に XP を実践した。Ron Jeffries がはじめての実践の中心者だった。

その後、Kent Beck は 1999 年に *"Extreme Programming Explained : Embrace Change"*[Beck99]という最初の XP 本を書き、世に XP を提示することになる。また、翌年に Ron は *"Extreme Programming Installed"*[Jeffries00]を書き、その実践方法を詳しく解説している。

また、Martin Fowler は *"Refactoring"*[Fowler+99]の著者であるが、この「リファクタリング」と呼ばれるデザインの再構築技術は、C3 プロジェクトでの XP 実践を通してはじめてその有効性が検証されたと言える。XP とリファクタリングの同時性は、両書籍の参考文献として互いに相互参照されている点からも興味深い。

XP の背景には、パターンコミュニティの影響が見逃せない。Kent Beck と Ward Cunningham は、ソフトウェアパターンの開祖となる一派(Hillside Group)であり、建築家 Christopher Alexander の「パターン言語」のアイデアを初めてソフトウェアに取り入れ、OOPSLA'87 で *"Using Pattern Languages for Object-Oriented Programs"* [Beck+87]として発表している。その後 Erich Gamma らによる *"Design Patterns"*[Gamma+95] が 1995 年に出版され、パターンはソフトウェアの世界でも脚光を浴びるようになる。

Ward Cunningham は、1995 年に *"EPISODES: A Pattern Language of Competitive Development"* [Cunningham95]という競争力のある開発チームに関するパターン言語を書いているが、ここには XP 的な考え方の萌芽がみられる。

XP シリーズ書籍紹介

2000 年、2001 年とあいついで XP 関連書籍が出版された。ここでは特に Addison Wesley Publishing Company の「XP シリーズ」を一通り紹介しておこう。このシリーズは、表紙に一冊ごとに印象的な色が使われており、色の名前で本を呼ぶことがある。どの本も Extreme の「X」の文字をフラッシュライトのようにデザインした装丁になっている (Kent

Beck によるとこの X は「鳩の糞」らしいが冗談の可能性が高い)。翻訳はすべてピアソン・エデュケーションから出版されている。

◆ “*Extreme Programming Explained: Embrace Change*”[Beck99] 1999 (白本)

著者：Kent Beck

邦訳：『XP エクストリーム・プログラミング入門 - ソフトウェア開発の究極の手法』

訳者：長瀬 嘉秀/永田 渉/飯塚 麻理香



XP のバイブルである。副題の“Embrace Change”(「変化ヲ抱擁セヨ」) は、Kent Beck によるいわば「XP 宣言」といえるだろう。序文を“*Design Patterns*”[Gamma+95]の Erich Gamma が書いている。

◆ “*Extreme Programming Installed*”[Jeffries00] 2000 (ピンク本)

著者：Ron Jeffries/Ann Anderson/Chet Hendrickson

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』

訳者：平鍋 健児/高嶋 優子/藤本 聖



XP の最初の実践者 Ron Jeffries によって書かれた詳細な実践の手引き。シリーズ中もっとも詳しく各プラクティスの実践方法を記述している。序文を Kent Beck が書いている。

◆ “*Planning Extreme Programming*” 2000 (グリーン本)

著者：Kent Beck/Martin Fowler

邦訳：『XP エクストリーム・プログラミング実行計画』

訳者：長瀬 嘉秀/飯塚 麻理香



XP の中でも、特にプランニングと見積について書かれており、Kent Beck と Martin Fowler の共著である。カードを使ったプランニングによって、いかに実践的かつ現実的に、精度の高い見積と確実な進捗を得ることができるかに焦点をあてている。序文を Peopleware の Tom DeMarco が書いている。

◆ “*Extreme Programming in Practice*” 2001 (オレンジ本)

著者：James W. Newkirk/Robert C. Martin

邦訳：『XP エクストリーム・プログラミング実践記編 - 開発現場からのレポート』

訳者：比嘉 康雄/平鍋 健児/高嶋 優子



米 ObjectMentor 社の Web サイトを、XP を使って実際に開発した際の実践レポート。Java で Servlet を使って開発をしていおり、ソースコードも添付されている。序文を Martin Fowler が書いている。

◆ “*Extreme Programming Examined*” 2001 (青本)



著者：Giancarlo Succi/Michele Marchesi

XP をめぐる 33 の論文集。シリーズ中、もっともぶ厚い。Martin Fowler

の”Is Design Dead”(『設計の終焉?』)も含まれる。

◆ “Extreme Programming Explored” 2001 (黄緑本)

著者：William C. Wake



オブジェクト指向プログラミング分野で名高い Wake による XP 本。テストファーストを中心に、XP プロセスが Java の例をふんだんに使って書かれている。序文を、Pragmatic Programmer (『達人プログラマー』) の Dave Thomas が書いている。

◆ “Extreme Programming Applied” 2001 (紫本)

著者：Ken Auer/Roy Miller



XP を導入する際に直面する困難にどうやって立ち向かったか、という先人の経験が詳しく書かれている。さまざまな重量級プロセスを試してきた著者たちが、XP に出会って確信した有効性を共有しようと、読者に熱く語りかける。序文を Ward Cunningham が書いている。

◆ “Questioning Extreme Programming” 2002 (黒本)

著者：Pete McBreen



ソフトウェア職人気質(Software Craftsmanship[McBreen01])の著者が、XP について。

XP の周辺 (アジャイルムーブメント)

XP の周辺として、アジャイルムーブメントについて紹介しよう。

XP のみがライトウェイトなプロセスとしての考え方を実現しているわけではない。「アジャイル宣言」(図6)は、アジャイルアライアンス(www.agilealliance.org)から出された宣言文[Fowler+01]である。

私たちは、

| | | |
|---------------|-------|-------------|
| プロセスとツールよりも | | 個人と対話に。 |
| 包括的なドキュメントよりも | | 動くソフトウェアに。 |
| 契約交渉よりも | | 顧客との協調に。 |
| 計画に沿うことよりも | | 変化に対応することに。 |

価値をおく。

図 5 アジャイル宣言(一部)

この宣言に参加しているのは , Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas で , XP をはじめ, Scrum, 達人プログラマ (Pragmatic Programmer) , DSDM, Crystal, Adaptive Software Development などといった開発手法の中心人物たちである . これらの手法は , より現場的 , 実践的な視点から現代的ソフトウェア開発に一つの光を投じているといえる .

XP もこの大きなアジャイルムーブメントの文脈における 1 つの方法論として見る事ができるのである .

XP のポジション

最後に , 図 5 に私が考える現在のオブジェクト指向技術の中での XP の位置付けを書いてみた .

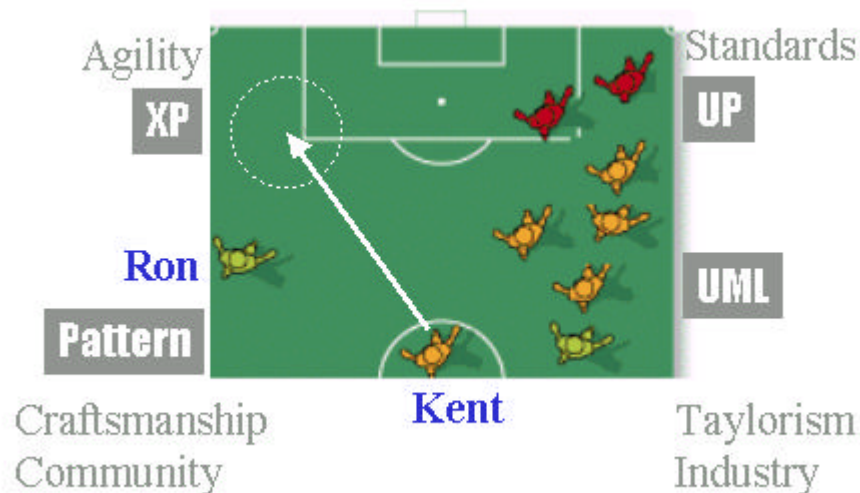


図 6 XP の位置付け

右翼には , UML(Unified Modeling Language) , UP(Unified Process)といった , どちらかといえば「標準」と呼ばれる技術 , 工学的にソフトウェアを捉えようとする技術がある . Kent Beck 流に言えば , これらは Taylorism (テイラー主義 - Frederick W. Taylor よる)

を踏襲している。すなわち、ソフトウェア開発を工業的、生産技術的に捉えており、トップダウンのアプローチとも言える。1995年から2000年に、オブジェクト指向界でもさまざまな「標準」とよばれる技術が現れてきた。

一方でまったく同時代的に、ソフトウェアパターンをはじめとする「現場のナレッジ」を尊重する動き（左翼）が起こった。このアプローチはボトムアップでコミュニティを形成し、コミュニケーションによってソフトウェアの現場を改善しようとしている。また、「匠の技」、「職人氣質(Pete McBreenのcraftsmanship[McBreen01])」のような従来の工学に乗らなかったような技術も伝承して行こうとしている。

XPはこの左翼の先端にあり、他のアジャイルムーブメントとともにAgility（拙速さ）を武器として現在もっとも注目されているソフトウェアプロセスである。Kent Beckは、1999年現在でぽっかり空いていたこのスペースを的確に見据え、そこにスループスを送りこんだと言えるだろう。確かにXPは極端だとも言われるが、その極端さのおかげで右翼と左翼の間のスペクトルがはっきりと衆目にさらされるようになった。XPはそのいささか誇張したオーバーアクションにこそ意義があるのだ。

後編へつづく

前編では、XPの概要とともに、XPが置かれている現在の文脈についても概説した。後編では、実際にXPを開発に取り入れた例を紹介していきたい。

参考文献

- [Beck99] Kent Beck, *"Extreme Programming Explained: Embrace Change"*, Addison Wesley Publishing Company, 1999(『XP エクストリーム・プログラミング入門 - ソフトウェア開発の究極の手法』,ピアソン・エデュケーション, 2000)
- [Beck+87] Kent Beck + Ward Cunningham, *"Using Pattern Languages for Object-Oriented Programs"*, <http://c2.com/doc/oopsla87.html>, 1987
- [Cunningham95] *"EPISODES: A Pattern Language of Competitive Development"*, <http://c2.com/ppr/episodes.html>, 1995 (『プログラムデザインのためのパターン言語 Pattern Languages of Program Design 選集』, ソフトバンクパブリッシング, 2001)
- [Gamma+95] Erich Gamma et. al, *"Design Patterns"*, Addison Wesley Publishing Company, 1995 (『オブジェクト指向における再利用のためのデザインパターン (改訂版)』, ソフトバンクパブリッシング, 1999)
- [Jeffries00] Ron Jeffries, *"Extreme Programming Installed"*, Addison Wesley Publishing Company, 2000 (『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』, ピ

アソン・エデュケーション, 2001)

[Jeffries01] Ron Jeffries, “*What is Extreme Programming?*”,
<http://www.xprogramming.com/xpmag/whatisxp.htm>, 2001 (『エクストリームプログラミングとは何か?』, <http://objectclub.esm.co.jp/eXtremeProgramming/whatisxp-j.html>, 2001)

[Cockburn+00] Alistair Cockburn, “*The Costs and Benefits of Pair Programming*”,
<http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF>, 2000

[Fowler+99] Martin Fowler et. al, “*Refactoring: Improving the Design of Existing* ”, Addison Wesley Publishing Company, 1999
(『リファクタリング - プログラムの体質改善テクニック』, ピアソン・エデュケーション, 2000)

[Fowler+01] Martin Fowler et. al, “*Manifesto for Agile Software Development*”,
<http://www.agilealliance.org/>, 2001

[Wells99] Don Wells, “*XP Map*”, <http://www.extremeprogramming.org/>, 1999

[McBreen01] Pete McBreen, “*Software Craftsmanship: The New Imperative*”, Addison Wesley Publishing Company, 2001

以上