XP(Extreme Programming): ソフトウェア開発プロセスの新潮流
～後編：XP 実践事例の紹介～

初出：情報処理 Vol.43, No.4 Apr.2002
(株)永和システムマネジメント
平鍋健児(hiranabe@esm.co.jp)

本記事は，『XP(Extreme Programming): ソフトウェア開発プロセスの新潮流』と題した記事の後編である．前編では，XP の概要およびその周辺を解説した．今回は，XP の実践レポートを紹介する．著者らは 2001 年 7 月，(株)永和システムマネジメントにおいて，医療患者管理を題材とした Web システム開発を XP で行った．ここでは，その結果を中心に考察を加えて報告しよう．

プロジェクトの概要

プロジェクトのコード名は「MPilot」(Medical Pilot Project)．位置付けとしては，以降の大規模実開発のためのプロトタイプ開発である．目的は，実開発へ向けてのプロジェクトメトリクスの収集，および XP 開発手法の評価である．

メンバーの構成は表１のとおり．

表 1 メンバー構成

メンバー名	開発経験	J2EE 経験	OO 経験	役割
SH	１０年以上	２年	１年	マネジャ、プログラマ
KH	１０年以上	１年	１０年	コーチ、トラッカ
YT	入社３年	２年	２年	プログラマ
ST	入社２年	１年	１年	プログラマ
KK	新人			プログラマ
AY	経験豊富			（擬似）顧客

このプロジェクトでは，「必ずしも全員のスキルが高くないプロジェクトにおいても XP

が適用できる」ことを示したかったため，できるだけ実際の現場でのバランスに近いメンバー構成とした．ただし，全員がなんらかの Java プログラムは書いたことがある．

参加するメンバーには，それぞれ役割がある．プログラマとマネジャは，通常の開発と同じと考えて良い．ただし，アーキテクトや設計者という役割はなく，プログラマ全員によって漸進的な設計が行われる．コーチは，チーム全体を見て回る役割である．プログラムしているペアを覗いては，「問題ない？」と聞いて回ったり，行き詰まっているペアの相談に乗ったりする．トラッカは，チームの開発速度を計測する．今回は，コーチがトラッカを兼任している．トラッカは見積どおりイテレーションが進んでいるかをウォッチし，メトリクスを収集する．メトリクスは報告のためだけではなく，イテレーションが進む中で見積に利用される．

また、仕様の決定に関しては顧客役（擬似顧客）に一名参加してもらった．擬似顧客は，この分野の業務知識をもっている．顧客にはフルタイムではなく，週に二日，作業場所を共にしてもうことにした．

実践期間は，2001 年 7 月の１ヶ月間．リリース計画では、開発要件をストーリーとして切り出し，各イテレーションにストーリーを割り当ててスケジュールした(表 2)．実践期間の都合で 1 イテレーション＝1 週間，リリースは全体(1 ヶ月)で１回のみである．

表 2　スケジュール

	2001 年 7 月																				
	2	3	4	5	6	9	10	11	12	13	16	17	18	19	23	24	25	26	27	30	31
リリース計画	計画																				
ＩＤ検索		第１イテレーション																			
条件検索		第１イテレーション																			
登録機能							第２イテレーション														
血縁関係												第3イテレーション									
トランザクション機能												第3イテレーション									
ソート機能																第4イテレーション					
ログイン機能																第4イテレーション					
まとめ																				全体	

開発環境

以下のようなソフトウェアツールを使って，システムを構成した．

- OS - Windows 98, 2000
- Web サーバ/Servlet エンジン - tomcat 3.2
- ビルドツール - ant1.3
- ユニットテスト - JUnit 3.7
- 受け入れテスト - HttpUnit 1.2.3
- ソース管理 - WinCVS 1.2
- コンパイラ - Java2 SDK 1.3
- クライアント - Web Browser(IE5.x)

今回の開発は，一般的な Servlet/JSP による Web アプリケーションである．データベース製品は利用せず，簡易な CSV 形式のファイルを用いて永続性を実現している．

ストーリー

開発要件は，個人の基本情報，患者情報や健診情報，職員情報の検索，表示，登録，変更ができる Web ベースのシステム開発である．画面例を図１に示す．

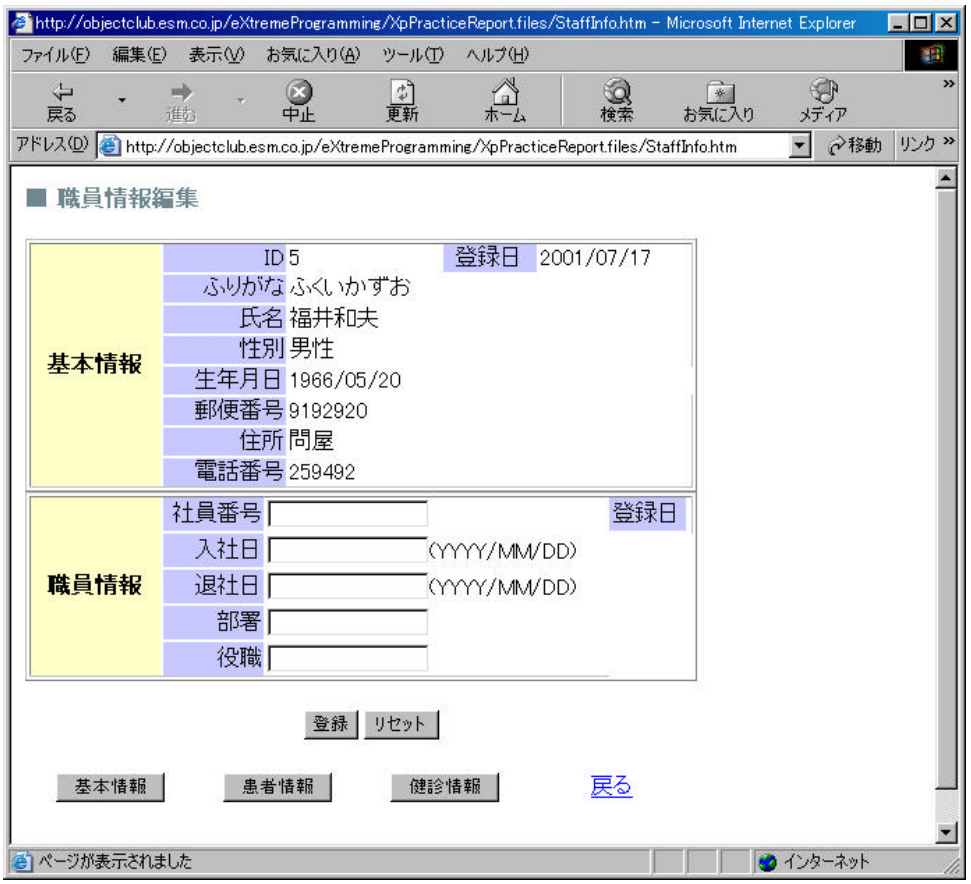


図　1　画面例

特殊機能として、血縁情報を追加して親子を順次検索できる機能も加えている．これらの開発要件をストーリーとして、以下のように切り出し，カードに書いた．ストーリーは,「誰にでも分かる仕様書」でないことに注意してほしい．顧客が書き，ストーリーの説明に参加した全員が，このカードを見れば会話のことを思い出すことができる点が重要である．また，このカードは壁に張り出され，開発中に疑問点があればカードを持って顧客とさらに会話をすることができる．

表３に最終的な全ストーリーをあげる．

表 3　全ストーリー

《ＩＤ検索機能》 ・人物のＩＤを入力して検索することができる。 ・データが見つかった場合は詳細情報画面にデータを表示する。 ・見つからなかった場合は検索画面に戻る。 ・見つからなかったというメッセージを表示する。
《条件検索機能》 ・条件を入力して検索することができる。 ・データが見つかった場合は検索結果を一覧画面に表示する。 ・検索条件は、ふりがな、氏名、性別、生年月日、電話番号、属性をＡＮＤ条件で検索する。 ・条件が１つも指定されない場合は全件表示とする。 ・一覧表示から詳細情報画面へ遷移する。
《登録機能》 ・登録画面からデータ（基本情報）を登録することができる。 ・患者情報、健診患者情報、職員情報を選択して登録できる。 ・登録画面（４画面）は任意に画面を切替えることができる。 ・登録実行時に、確認画面を表示する。 ・各種データはバージョン管理する。
《血縁関係の追加》 ・基本情報に父親、母親、配偶者の情報を追加する。 ・詳細情報画面、登録画面（基本情報)、登録確認画面に血縁関係を追加する。
《トランザクション機能》 ・各種情報のトランザクションを一元管理して、情報矛盾をガードする。
《ソート表示機能（一覧画面)》 ・一覧画面にソート表示を追加する。 ・ソート条件は、ＩＤ，ふりがな，生年月日とする。
《ログイン機能》 ・ログイン画面を作成して、セキュリティ機能を追加する。 ・ログインなしによる各種参照画面、登録画面へのアクセスをガードする。

リリース計画

リリース計画では，ストーリーを全員で見積もり，それをイテレーションのスケジュールに分割する．前述のストーリーは最終的なものであり，最初に出たストーリーは,《検索機能》,《登録機能》,《血縁検索機能》の３つのみだった.

私たちはおのおののストーリーを「2 理想週(10 理想日)」と見積もった．理想週とは，自分が一人でプログラミングに没頭したときにどのくらいの期間が掛かるか，という値である．実際に掛かる工数は，この理想週に負荷係数を掛けた値となる．私たちは５名いたの

で，1 週間の実際の保有工数は 5 人週である．負荷係数として 2.5 を採用[1]し，1 週間の作業量を，5/2.5 = 2 理想週分とした．すなわち，ちょうど今回の 1 ストーリーを 1 週間でこなす計算となる．

理想週や理想日による見積では，その絶対値に大きな意味を置かない．とくに最初のイテレーションではあてずっぽうだ．そのため，見積を「ポイント」という無色の単位で呼ぶことが多い．イテレーションが進むにつれて相対的な見積が安定し，１イテレーションで何ポイント開発できたか（これを「プロジェクト速度」とよぶ）という値が収集できる．これを次のイテレーションで開発できる量として計画を進める．

このリリース計画には，ストーリーカードの記述も含めて 1 日を費やした．

イテレーション計画

イテレーション毎に，イテレーション計画を行う．私たちは 1 イテレーションを 1 週間で行ったので，毎週月曜朝の最初の活動がこのイテレーション計画であった．例として，第 1 イテレーションの計画を見ていこう．

まず，イテレーションの最初にイテレーション計画を立てる．第１イテレーションへの入力は，《検索機能》ストーリーである．イテレーション計画のミーティング(全員参加の 2 時間程度の会議)を進めるなかで，このストーリーは 2 つに分割した方が分かりやすいという結論になった．すなわち，《ID 検索機能》と《条件検索機能》である．

私たちは，この２つの機能を実装するためのタスクをブレークダウンした．全員でブレインストーミングを行いながら，模造紙を壁に貼ってタスクを書き出す(図 2)．タスクは８つであり，例えば「コーディング規約の配布」や「CVS 環境の整備」などもこのイテレーションには入っている．

タスクが一通り出たら，それに各自がサインアップする．図 2 の，SH や AY などのイニシャルが，開発者各自のサインである．サインアップの際，そのタスクの見積も同時に行う．タスクの横のカッコ内に書かれた 1.5 や 0.5 という数字がその見積である．見積の単位は理想日（５理想日＝１理想週）である．サインアップした人は，そのタスクの完了責任者となる．各自，合計の見積がオーバーコミットとならないように注意する．一週間は 5

[1]一般には，ペアプログラミングやミーティングに費やす時間があるため，負荷係数は２以下にはならない．仮に割り込みが全くなく，かつ，見積が正確だったとした場合でも，自分が使える時間のうち半分は他のメンバーからペアプログラミングに借り出されることになるからだ．Ron Jeffries は 3.0 から始めることを推奨している[Jeffries00].

日であるが，負荷係数が 2.5 であるから，一人合計 2.0 以上はコミットしないようにする．

第１イテレーション(7/3 - 7/9) 計画

1．CVS 環境整備 (0.5)
　SH - 0.5
2．コーディング規約
3．画面設計，遷移
　AY(顧客)に任せる
4．人物・患者等のモデルクラス作成 (1.0)
　YT - 1.0
5．モデルストア（簡易 CSV ファイル DB メカニズム）の仕組み (1.0)
　KH - 1.0
6．検索サーブレット (2.0)
　SH - 1.0
　ST - 1.0
7．表示用 JSP (1.5)
　YT - 0.5
　ST - 0.5
　KK - 0.5
8．HTML 検索フォームファイル (1.0)
　KK - 1.0

図 2　第１イテレーションのタスク

開発の実際

毎朝は，スタンダップミーティングで始まる．ここでは，自分の進行具合，問題点，を出し合う．時間は 15 分を目安にすべきである(立ったままできる時間)．時として，細かいコードの議論になってしまうこともあるが，問題の粒度によってはマネジャやコーチが適宜「これに関しては後で KH と YT で話そう」などと話を区切る．

大きな見こみ違いや技術的な障害がある場合，その分野に詳しい他のメンバーに意見を求めたり，困っている人とペアを組むように勧めたりするのがマネジャの仕事である．

開発はペアプログラミングによって行われる．ペアリングの基本は，「ペアを頼まれたら断れない」である．タスクへのサインアップは各自が行うが，ペアのパートナーは，各自がその都度空いている人やその分野に強い人にお願いする．また，スタンダップミーティング時にマネジャやコーチがペアを推薦することもある．

今回の計測では，全作業時間の８割がペアプログラミングに費やされている（テストも含めて）．図 3 がペアプログラミングの様子だ．

図 3　ペアプログラミング

ペアがタスクを終了すると，イテレーション計画の模造紙の該当タスクに横線を入れて，終了を宣言する(図４).

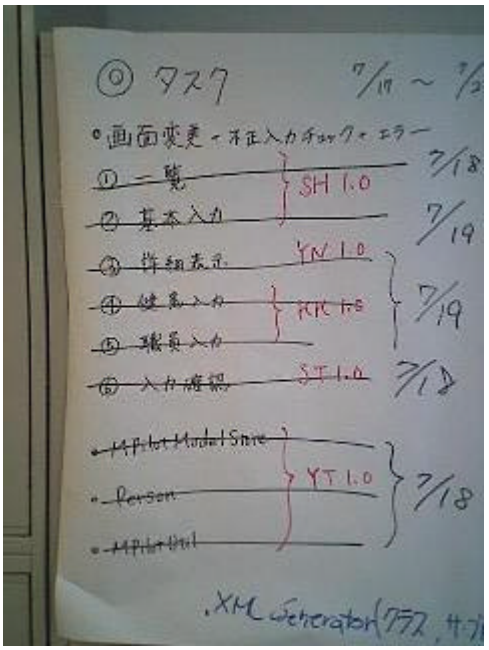


図 4　終了したイテレーション

イテレーション毎のタスク計画の他に，キーとなる設計図を壁に貼った．この開発では，全体の画面遷移を示す図がキーとなった(図５)．開発中に設計に困ると，この図を見ながらペアで話をすることが多かった.

図 5　壁に貼られた画面遷移

顧客の役割

顧客は，週に 2 日ではあったが開発チームと席を並べて質問に答えた．

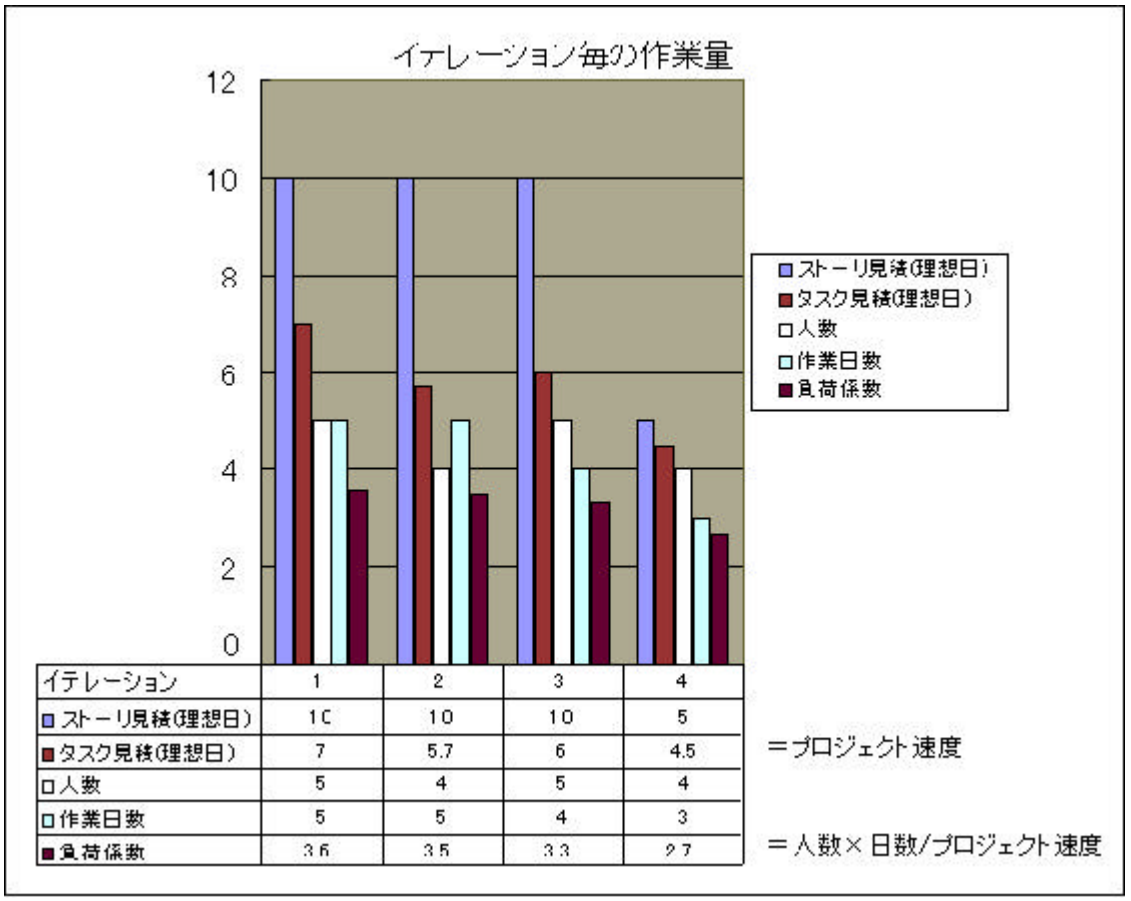
今回は，幸運にも顧客はプログラムを書くことができた．顧客は HttpUnit を使って，HTTP リクエスト(入力フォーム)と HTTP レスポンス(結果の HTML) をチェックするプログラムを書いた．テストプログラムは，各イテレーションの終了時にはすべて通過させる必要がある．

おそらく，現実には顧客にプログラムを書くことを求めるのは難しいケースがほとんどだろう．その場合には，顧客がテストを書くのをサポートするテスターを開発側から準備する必要があると思われる．

プロジェクトの結果

トラッカが収集したメトリクスは，イテレーション計画に書きこまれた見積と，その実績がすべてだ．ただし，既存のプロジェクトとの比較のために，便宜的にコードの行数やクラス数も収集した．

図６に，イテレーション毎の作業量を示す．「プロジェクト速度」とは，そのイテレーションで何理想日分の作業が消化できたかを示す．結果としての負荷係数は，実際の保有工数をこの値で割ったものとなる．



イテレーション	1	2	3	4
ストーリ見積(理想日)	10	10	10	5
タスク見積(理想日)	7	5.7	6	4.5
人数	5	4	5	4
作業日数	5	5	4	3
負荷係数	3.6	3.5	3.3	2.7

＝プロジェクト速度

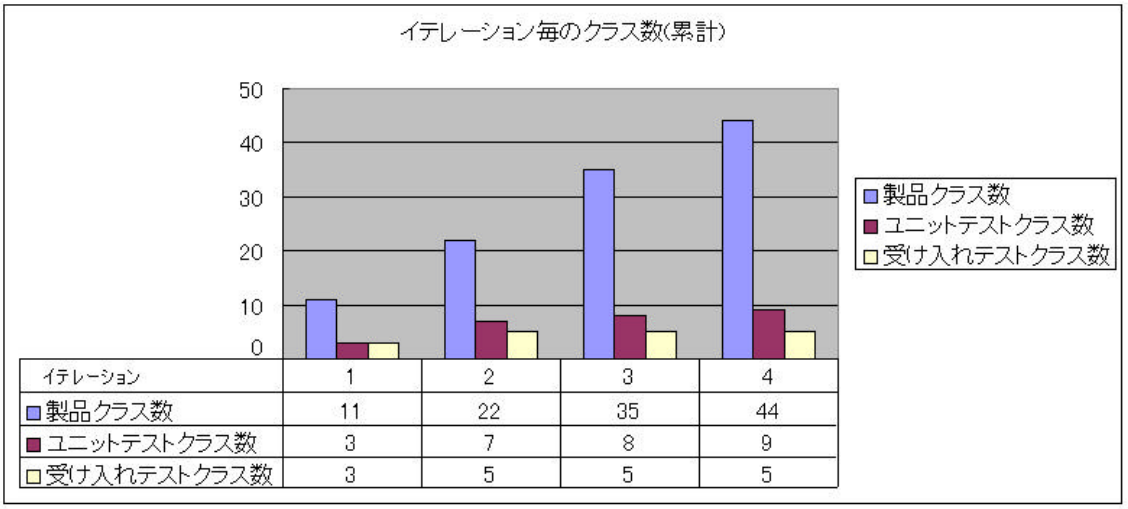
＝人数×日数/プロジェクト速度

図 6　イテレーション毎の作業量

プロジェクト速度が一定していない原因には，イテレーション毎のメンバーの増減，利用可能日数，チーム能力の向上，見積能力の向上などが考えられる．また，ストーリー見積とタスク見積合計(プロジェクト速度)が一致しないのは，タスクにブレークダウンすることで問題が思ったより複雑もしくは簡単であることが分かったためである．負荷係数は，見積時は 2.5 で始めた．しかし，人数や日程の変化で，第 1 イテレーションの結果としての実質値は 3.6 であった．また，最終イテレーションでは 2.7 になった．この数値はプロジェクト速度を変化させる上記要因で変化するが，結果的には負荷係数が徐々に下がったことになる(効率が上がったことを意味する).

図７に，イテレーション毎のクラス数の累計を示す．クラス数には，Servlet/JSP/HTML のすべてを含む．ユニットテストは，モデルクラス(Servlet や JSP 以外の JavaBeans)に

しか用意しなかった(できなかった).



イテレーション	1	2	3	4
製品クラス数	11	22	35	44
ユニットテストクラス数	3	7	8	9
受け入れテストクラス数	3	5	5	5

図 7　イテレーション毎のクラス数（累計)

Servlet や JSP もテストする予定ではあったが，なかなかうまく行かず，途中で「ロジックは Servlet や JSP には入れず，モデルに追い出せ」という方針を採った.

一般に，ソフトウェア設計の方針にはいくつか考えられる．例えば，「再利用性を高める設計」,「保守性を高める設計」,「パフォーマンスを高める設計」などの視点があるだろう．しかし，テスト駆動のプロセスである XP では「テストが容易な設計」に行きつく．これは今回発見した XP の副次的な効果であると思う． Ron Jeffries は,「シンプルな設計」の条件として,

- すべてのテストをパスし,
- 意図（何をしたいのか）が明確に表現されており,
- どんなことも一度，ただ一度だけ記述されており,
- 上記を満たしながらクラスとメソッドの数が最小である.

をあげている[Jeffries00]．特に，テストの容易性は環境（利用できるツールなど）によって左右されるものであるから，この設計方針は環境に依存する性質を併せ持っていると考えられる．直感的には，設計が環境に過度に依存することに疑問を持つ方も多いと思う．しかし，この環境（ビジネス価値やツールも含めて）に対する適応能力こそが「変化を抱擁」する XP の本質であることを考えれば,「テストが容易な設計」は XP の考え方とよくマッチする.

また，今回受け入れテストは，全体をカバーするだけの工数が顧客に足らなかった．受け入れテストは明らかに不足している．

図 8 には，各イテレーションでのテストパス率を示す．ユニットテストは必ず 100%でなくてはならない．

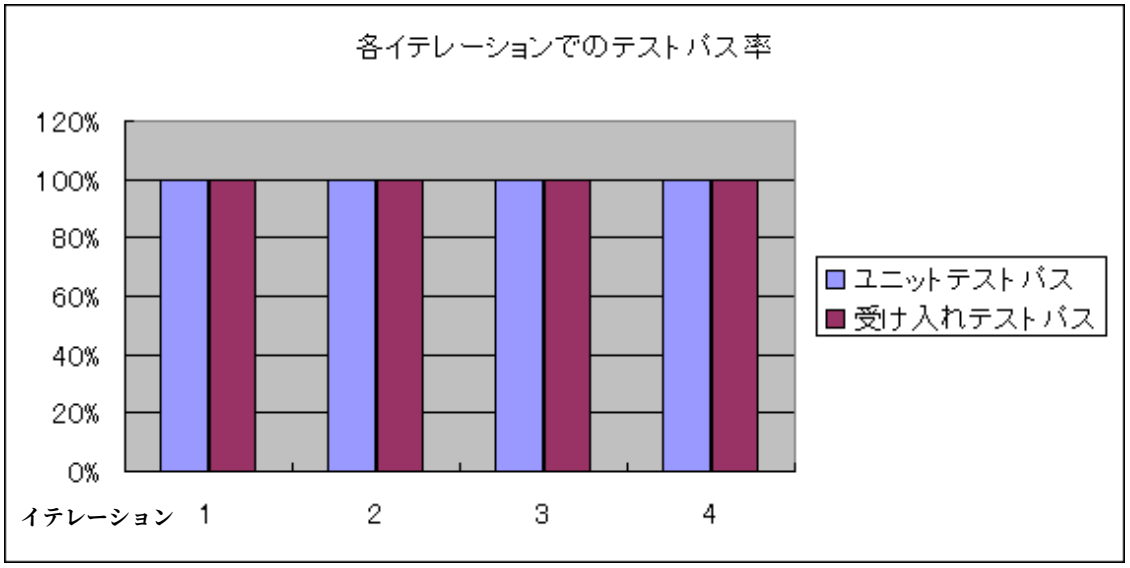


図 8　各イテレーションでのテストパス率

表４には，プロジェクト全体のメトリクスを示す．このメトリクスは XP には本質的ではないが，過去のプロジェクトと比較するために便宜的に採取したものである．

表 4　全体のメトリクス

実作業人日	77
製品クラス数	44
製品 LOC	7314
テスト LOC	1655
全 LOC	8969
製品 LOC/人月	1899
全 LOC/人月	2329

(LOC = Line Of Code　コメント・空行ぬきの行数)

プラクティスの実際

今回のプロジェクトでは，XP のプラクティスを次のように実践した．実際の 14 のプラクティス毎に，今回の適用方法を示す．

◆ 計画ゲーム

（擬似）顧客を交えて，ストーリーカードを利用して行った．リリース計画は一回のみ．イテレーション計画は４回行った．

◆ 小さなリリース

リリースは一回のみ．

◆ メタファ

今回行わず（思いつかなかった）．ただし，JSP Model2 の MVC アーキテクチャが，全体構造を説明する機能を果たしたと思う．

◆ シンプルデザイン

徹した．できるだけ明日への設計をしなかった．次のイテレーションで入ってくる予定のストーリーについても敢えて無視して，そのイテレーション限りの設計を進めた．

◆ テスティング

ユニットテストは，モデルクラスについてほぼ全クラス行った． Servlet/JSP については，行えなかった． 受け入れテストは HttpUnit を用いて行った．

◆ リファクタリング

適宜行った． リファクタリング方法の教育，伝播にはペアプログラミングが役だった．

◆ オープンワークスペース

席の並び替えを行い，できるだけ近くで作業をするようにした． オープンなスペースもあった． ホワイトボードはコミュニケーションの必須アイテムであった．

◆ ペアプログラミング

徹底して行った．すべての製品コードはペアによって書かれた．

◆ 共同所有権

CVS を利用し，全員が自由に変更した． CVS はグループ作業には必須であった．

◆ 継続的インテグレーション

今回は統合マシンによる自動ビルドや自動テストは行えなかった．

◆ 週４０時間

結果的に達成できた．

◆ オンサイト顧客

擬似顧客をオンサイトに置いた．ただし，週に２日のみ．

◆ コーディング標準

オブジェクト倶楽部バージョンのコーディング標準[Hiranabe00]を採用した．

◆ 日ごとのデータベース移行

データベース製品は利用しなかったが，常に最新スキーマを用いた．

考察

今回の１ヶ月の実験によって，多くの考察が得られた．XP は本や記事を読んだだけではなかなかコツが掴めない．特に現場でリアルタイムに開発のリズムを感じることで得られることも多い．以下にその幾つかを紹介したい．

◆ 知識の伝搬

今回の最大の発見は，ペアプログラミングによる知識の伝搬の速さである．ツールにしろプロセスにしろ，今回ははじめての要素だらけだった．今までの開発であれば，ツールの利用方法のドキュメントをまとめたりしていたのだが，実はそんなことは肝要ではなく，ペアになって作業を通して理解を伝え合う方がよっぽど効率的であることが分かった．

◆ プランニング

XP プロジェクトを軌道に乗せる上でもっとも重要なのは，イテレーション計画だと感じた．特に，このミーティングの司会を首尾よく行うことには，経験とスキルと信頼が必要．また，XP に対する批判として，「スキルの高いチームであることが必要条件ではないのか」というものがあるが，今回は全体的に高スキルとは言えないチーム構成である．しかし，「予測可能な進捗」(速い進捗ではない)を得ることができ，見積の精度は非常に高かったといえる．これは，マネジャにとって低リスクというメリットがある．

◆　先行型設計と進化型設計

データベースとなるクラスの設計を途中で大きく変更した．実際，途中で大きなリファクタリングが起こり，そのために一日以上がつぶされている．最初にしっかりとモデリングを行えば，この作業は不要だったかもしれないが，本当にそうだったかは，誰にもわからない．私たちは必要になった時点でそれを行った，ということは言える．ただ，あまり頻繁にそれが起こるのは，正直辛いだろうという印象は持った．

◆　マネジャとコーチの役割

マネジャとコーチは，障害物をどけてまわることが一番大事な役割．つまづいているペアがいたら，相談にのって，つまずいている小さな石をどけること．これが全体を前にすすめるコツ．

◆　中心的プログラマ

自然発生的に，リーダーシップをとり，全体にアドバイスを与えることができる中心的プログラマが出現した．彼女は全体のバランスを見ながら構造に関しての一貫性をシステム全体に浸透させた．もしかしたら，XP でアーキテクチャが自然発生するように，「アーキテクト」という役割も自然発生するのでは，と考えている．

◆　JUnit

JUnit によるユニットテストは非常にうまく機能した．最後までモデルのユニットテストは 100% をキープした．ただ，Servlet/JSP のよいユニットテストの方法を発見できなかったため，Servlt/JSP からはできる限りロジックをモデルのコードに追い出す，という戦略を取った．

◆　Ant/CVS

この組合せでの構成管理は，大成功と言える．第 1 イテレーションで全員が使い方をマスターしたことは，ペアプログラミングの効用の１つである．

◆　テストファースト

残念ながら，あまり浸透しなかった．アイディアは前もって布教したのだが，やはり，製品コードを先に書き，テストする，という方が「楽」に感じたようだった．先にテストを書く，というのには，面倒な印象があり，結果としては定着しなかった．熱心にテストファーストを行うプログラマが一人いれば，結果は違ったと思われる．

◆　オンサイト顧客

「そこに顧客がいる」というのは，いつでも質問でき，曖昧な仮定のままでプログラミ

ングを行わないでよいという利点がある．通常の方法で開発を経験しているプログラマからも，この点は大きなメリットとして指摘された(現実問題としてできるか，というのは置いておいて)．また，顧客がテストを決めるとなると，顧客の負荷もかなり大きいことは事実．今回は時間がなく，受け入れテストの数が足らないことが残念．

◆　新人育成の場としての XP

今回は，このプロジェクトには新入社員の OJT の場としての役割もあった．私自身は，今回の結果に非常に満足している．新人が仕事のやり方，言語，ツール，進捗報告などソフトウェア開発にひととおり必要な能力を，複数の人とのコミュニケーションの中で獲得することができたからだ．新人の彼女は常にタスクをコミットしていたし，通常の OJT にありがちな，「放っておかれる」という状況がなかった．

◆　温室プロジェクト

今回のプロジェクトは，契約，予算，納期，実際の顧客などという実プロジェクトではクリティカルな要素がほとんどない，ある意味で温室プロジェクトであることは告白しておくべきだろう．全体が和気あいあいとした雰囲気の中でできたことは，XP という手法以上に，このことが大きいと感じている．

XP のススメ

本記事では，当社で行われた XP プロジェクトの報告を行った．

XP に関する文献は，2002 年現在でほぼ出尽くした感がある．実際に XP を現場に適応していいくためには，

◆　実際の現場での適応事例報告
◆　それをふまえた日本での XP のローカライゼーション

がますます必要になってくると思われる．XP の情報源としては，日本の XP ポータルである，

http://ObjectClub.esm.co.jp/eXtremeProgramming/

を参照するとよい．この中には，XP-jp メーリングリストへの参加方法も含まれる．また，日本 XP ユーザグループ(XPJUG)では，実践報告や勉強会などが行われている．

http://xp.medinfo.m.ehime-u.ac.jp/

多くの方が XP コミュニティーへ参加し発言することで，日本での現実的な XP の適用が進み，現場で開発者が活き活きとプログラミングできる環境が浸透することを望んでいる．

参考文献

[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）

[Hiranabe00] Kenji Hiranabe, "Java **コーディング標準** - **オブジェクト倶楽部バージョン**",http://ObjectClub.esm.co.jp/eXtremeProgramming/CodingStd.doc, 2000

以上