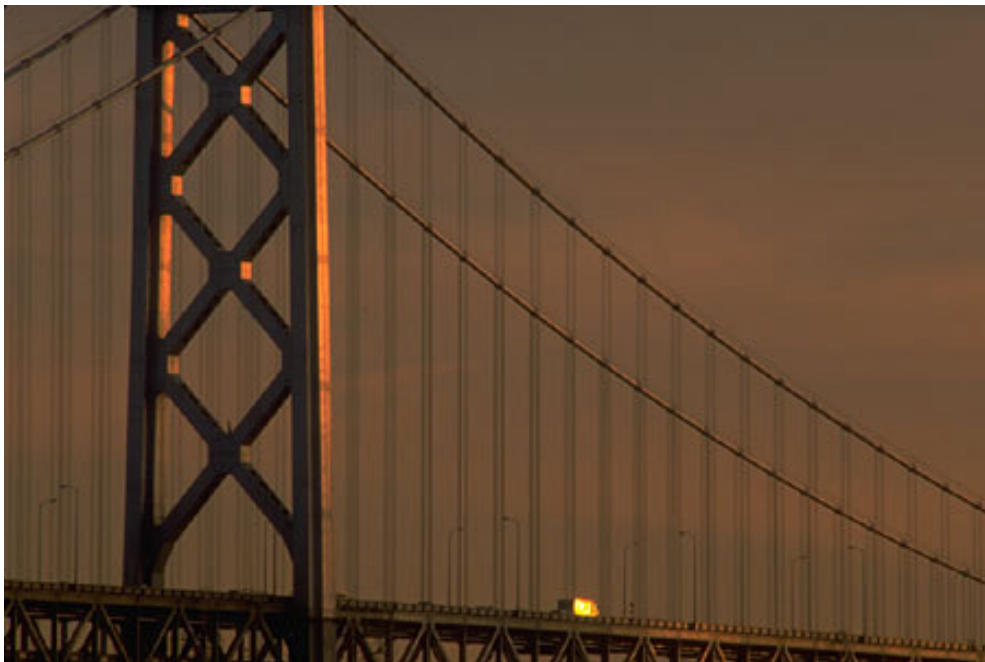




## eXtreme Programmingへの誘い



日本コンピューター・システム株式会社

<http://www.ncs.co.jp>



## はじめに

eXtreme Programming(以降、XP と省略)については、すでに多くの書籍で語られている。そのため、筆者の独断的な観点からXPについて述べることにする。それは、多くの方々からご批判を仰ぐことになるだろう。だからと言う訳でないが、先にお断りしておきたい。一人の技術者の戯言として見て頂きたい。

かなり乱雑な文章にはなっているが、本当に XP を理解し、実践していただきたいためにあえてこのようになってしまった。

また、今回は、サイトに掲載した文章をほぼそのまま文書化している。これは、そのときの「思い」を伝えたいからである。その「思い」を受け取っていただければ大変喜ばしいことである。

XPは決して容易い開発手法ではない、むしろかなり難しい開発手法であろう。  
これは、しっかりとした哲学としっかりとしたソフトウェアプロセスが隠れているからであろう。

本書が、皆様にとって1つの手引きとなれば幸いである。

平成 14 年 11 月 13 日

新保康夫  
(サイトを文書化に際して)

## 目 次

第1章 eXtreme Programming の罫.....	1
1. 1. XPは魔法の言葉 .....	1
1. 2. プログラマの意味 .....	1
1. 3. プログラマ復権.....	2
1. 4. それは、スーパープログラマ .....	3
1. 5. 匠の世界、その伝承 .....	3
1. 6. XPの呪縛.....	4
第2章 4+4+5+12 ～ 温故知新 ～ .....	5
2. 1. XP、お馴染みの言葉 .....	5
2. 2. 温故知新.....	6
2. 3. XP の ABC .....	6
第3章 4の数字 ～ 変数と値 ～.....	7
3. 1. 4つの変数 .....	7
3. 2. 4つの値 .....	8
3. 2. 1. コミュニケーション.....	8
3. 2. 2. シンプル .....	8
3. 2. 3. フィードバック.....	9
3. 2. 4. 勇気.....	10
第4章 基本原則 ～ 本当に当たり前と言えますか ～.....	11
4. 1. 瞬時のフィードバック.....	11
4. 2. シンプルの採用.....	11
4. 3. インクリメンタルな変更 .....	12
4. 4. 変化を取り入れる .....	12
4. 5. 質の高い作業 .....	13
第5章 12のプラクティス ～ 深海の底に辿り着けるか ～.....	14
5. 1. 短期リリース ～変化するという事実～ .....	14
5. 1. 1. 顧客サイドが重要.....	14
5. 1. 2. 顧客は変化する.....	14
5. 2. テスティング ～発想の転換～ .....	15

---

5. 2. 1. テストは Requirement を保証する.....	15
5. 2. 2. Requirement を保証するためのテスト項目を列挙する.....	16
5. 2. 3. 発想の転換 .....	16
5. 3. ペアプログラミング ～技の激突、そして融合～ .....	17
5. 3. 1. 生産性と品質.....	17
5. 3. 2. リスクマネジメント.....	17
5. 3. 3. 技の激突、達人への道.....	18
5. 4. 40 時間労働 ～本当の生産性～ .....	19
5. 5. コーディング規約 ～見えないドキュメント～ .....	20
5. 5. 1. 内部のドキュメントがない訳.....	20
5. 5. 2. 本来、ドキュメントを作成する人.....	20
第6章 明日を担うもの ～ 哲学と両輪の輪 ～ .....	22
6. 1. XP は哲学か .....	22
6. 2. ソフトウェアプロセスとの関係 .....	22
6. 3. 明日を担うもの .....	23
付録 参考文献.....	25

## 第1章 eXtreme Programming の罫



### 1. 1. XPは魔法の言葉



ここ数年間、システムの開発方法は、パンドラの箱を開けてしまったのか大きなパラダイムシフトの中に入ってしまった。それは、急速な社会変化と著しい情報技術革新との波が情報システムの開発を、闇闇の世界へ引きずり込んでしまったからだ。

それまで軽視されていたオブジェクト指向開発が、その闇闇に放つ光となってシステム開発を救う、溺れる者にとっての藁のごとく情報技術者をその方向に導いた。その中で、ひとときわ鋭く閃光を放つのがXPである。

XP—それは情報技術者にとって魔法の言葉のように入ってくる。とても魅惑的な言葉で。「実装ありき。まずは、実装しよう。」「コードが設計書。仕様書なんていらぬ。」と。(XPについての見識ある方なら、この大きな誤解はすぐにわかるであろう。)この魔法の言葉が多くの情報技術者を惑わし、闇雲に否定的な情報技術者や失敗する情報技術者を産み出した。

XPは魔法の言葉ではない。オブジェクト指向技術を十分理解できた人達、長年オブジェクト指向技術による開発に従事してきた人達にとってのひとつの開発方法なのである。

### 1. 2. プログラマの意味



XPについて述べるときにいつも気になることがある。それは、「プログラマ」である。情報システム開発者たちを「システムエンジニア」と言い、「プログラマ」は何かその下の技術者という感じがする。これは私一人が感じていることだけなら良いのだから。しかし、そうではないように見える。

ところで、システムエンジニアは何者なのだ。分析者ならアナリスト、情報環境(ネットワークやデータベース環境など)を構築するのはアーキテクト、そして実際に実装するものに携わるのがプログラマである。システムエンジニアという名称など情報システム開発においては出てこないのだ。この言葉こそ、日本におけるソフトウェア開発の失われた10年(日本経済と同様にソ

ソフトウェアにおいても実はあると私は考える)のひとつの象徴なのである。

今、再び、プログラマの意味を考えたい。プログラマとは、実装におけるPLAN-DO-CHECK-ACTIONを行うプロフェッショナルな人達ではないのか。とすれば、本当にプログラマと胸を張って言える人はどれくらいいるのだろうか。

### 1. 3. プログラマ復権

ユーザが本当に必要とするシステムはどうすればうまく作れるのだろうか。実は、その答えは意外と簡単なのだ。ユーザがユーザ自身でその思いを直接プログラマに伝えてシステムを作ればよいのである。



そのときに必要なのは、その思いを理解できるプログラマである。

意味のない設計書やユーザの思いを正しく伝えられない設計書しかプログラマに渡せないのなら、ユーザの思いを理解し、実装におけるPLAN-DO-CHECK-ACTIONができるプログラマにユーザが直接伝えた方が良いではないのか。

このことは、プログラマがシステム開発において低いレベルの技術者ではなく、そのフェーズにおける高度な技術を要する情報技術者であることを示している。

確かに、ユーザの思いを正しく伝える設計書を書ける情報技術者は多くいるであろう。しかし、本当に少ないページで設計書が書かれているだろうか。小さなシステムでも、数百ページのドキュメントになっているのではないだろうか。果たしてそれはすぐに、簡単に目を通して、理解できるドキュメントと言えるのだろうか。実は、プログラマが書くドキュメントを書いているだけではないだろうか。

XPを理解していく過程で、私は、これはプログラマの地位と権利を戻そうとしているように思えてくるのである。まさに、『プログラマ復権』である。

XPは、プログラマがユーザの思いを自分達の力で実現できるということ、そして、それはプログラマにとっての楽しみであるということを教えてくれているように見えるのだ。

#### 1. 4. それは、スーパープログラマ

これは私自身だけが感じているのだから、プログラマがユーザの思いを実現するシステムをすべて最初から最後まで成し遂げているように見られる。これは屋台方式という一人工程がソフトウェア開発の世界で実現する手法としてXPになるのではと考えるからである。

この方式は昔からあるのである。そう、皆さんも良く知っている「職人」と呼ばれる人達である。職人というと、頑固一徹、技術が徒弟制度でしか伝わらないなどがすぐに思ってしまう。

しかし、そのような職人とは明らかに異なると思われる。ということは、新しい職人、つまり、New 職人である。そしてプログラマで言うならば、それはスーパープログラマと呼ばれる人達であろう。



だとすれば、XPはスーパープログラマを必要としているのか。スーパープログラマであれば私自身が感じていることは容易に解決がつくのだ。しかしそれでは、XPはごく一部のプログラマしかできないことになってしまう。

#### 1. 5. 匠の世界、その伝承

ユーザの要求を聞いた時点で、パソコンのキーボードからテストコードと実際のコードが直接打鍵できるプログラマは、まさに、達人であり匠である。そのような世界が匠の世界と言える。その匠の世界に多くのプログラマが到達すれば、システムは容易に作成されることになる。「そんなことは不可能。」と多くの人は声を大にして叫ぶだろう。



確かに、まったく同じことをすることは不可能だが、形を変えてそこに近づくことは可能である。単に匠であるスーパープログラマがその技術を他のプログラマに伝えるだけでは、長い時間を要することになるであろう。伝えることも大事だが、スーパープログラマである匠がどのように思考していくのか、どのように作業をすすめていくのか—そのプロセスを分析し、そのプロセスを他のプログラマでも可能な形に進めていけば、その技術はより早く、より多くの人に伝承し、匠の世界に近い世界を体験することは可能であろう。

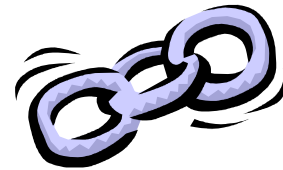
XPは匠の世界に近づける開発方法と言えよう。誰でもとは言わないが、多くのプログラマがスーパープログラマとして同じように振舞えるためにXPは必要なのだ。

## 1. 6. XPの呪縛

プログラミングすることは楽しいことであるということを知っている人にとって、プログラマとしての誇りや匠の世界を片隅に置き忘れてきたことをXPは後悔させるものである。私自身、もはや錆付いた道具ではプログラミングすることもできない。XPはその慕情を懐かしむには十分すぎるほどのものである。

とは言え、一人でも多くの方々に正しくXPを理解し、XPを実践してもらえるよう語り部として伝えることはできるのである。心強いことに、XPという魅力的な罠にはまった情報技術者が正しいXPを実践しているのも事実である。

どうやら、私自身もXPの罠にはまってしまい、その呪縛から逃れられないようだ。

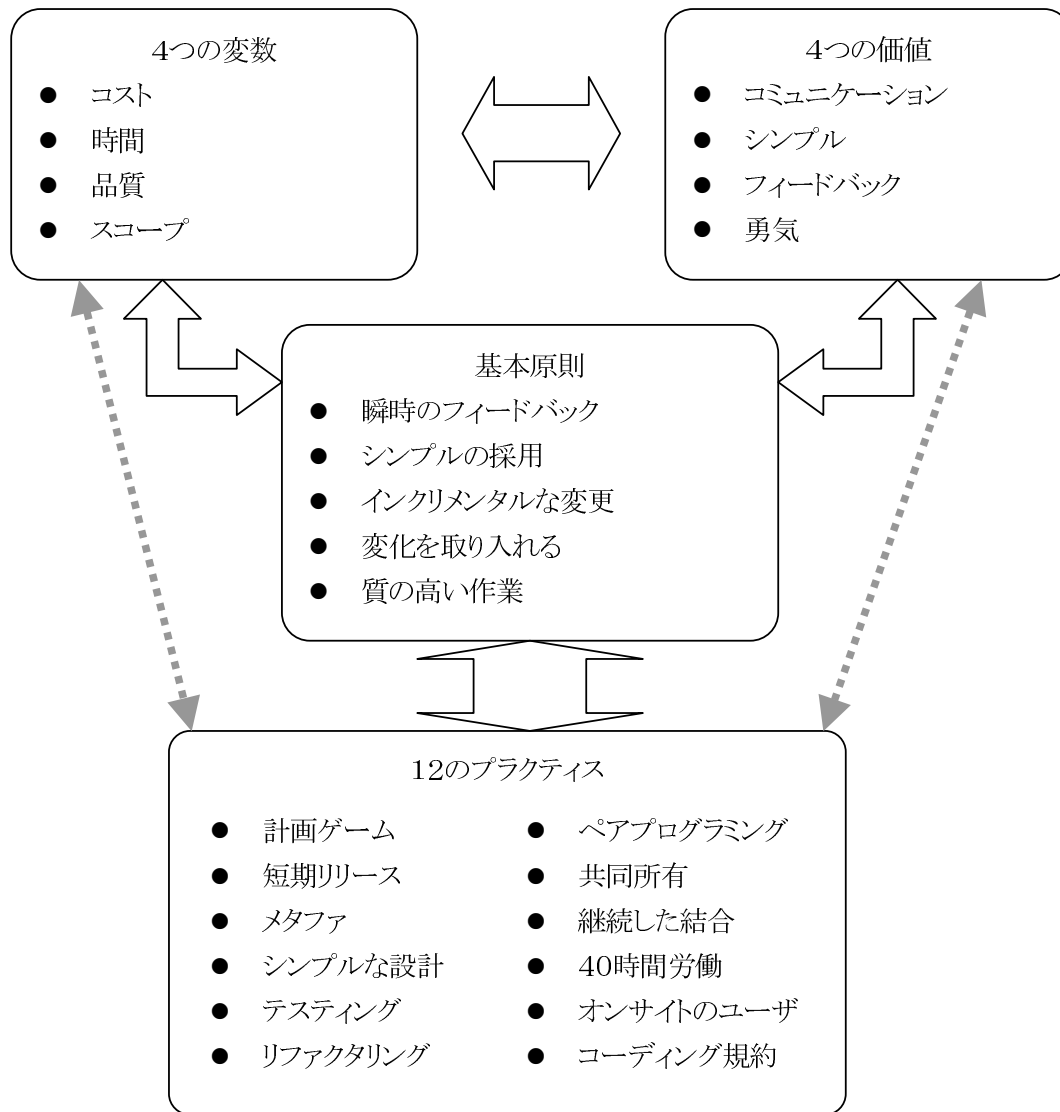


## 第2章 4+4+5+12 ～ 温故知新 ～



### 2. 1. XP、お馴染みの言葉

XPを語る上でやはり、お馴染みの言葉を挙げなければならない。そう言えば、この章のタイトルはすぐに察しがつくであろう。4つの変数、4つの価値、基本原則、そして12のプラクティス（最近では、13のプラクティスになっていて、一部プラクティスの名称やそのものが替わっている）である。ということで、これらの言葉をまずは列挙しておこう。



## 2. 2. 温故知新

皆さんは「XP で列挙した項目に新しいものは何一つない」と言われるであろう。確かに、そうである。目新しさは一つもない。しかも、重要なことばかりである。しかし、皆さんは本当にこれを理解し、実践しているのだろうか。私は大いに疑問の声を投げかける。

XP の項目を語るたびに私は次の言葉を思い出す。「温故知新」。意味は語る必要もないであろう。まさしく、XP は「温故知新」である。

## 2. 3. XP の ABC

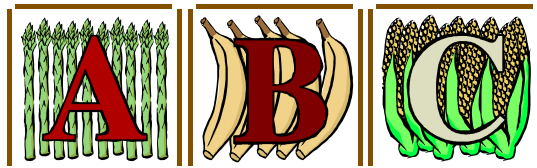
では、何が開発手法と言われるのであろうか。実は、これらの項目を ABC とするからこそ、XP は開発手法と言われるのである。

その ABC とは、

**A:Architect**  
**B:Bind**  
**C:Coordinate**

である。

もし、この言葉で皆さんの中で XP って何か薄ぼんやりと見えてきた人がおられれば、もう、あなたも XP の信者となりはてる。そして、納得し、実践してみたいと考えているならば、XP 開眼の境地にはいり、巷に徘徊する XP の達人たちの仲間となる。



## 第3章 4の数字 ～ 変数と価値 ～



### 3.1. 4つの変数

システムを開発する上で、プロジェクト管理者、ソフトウェア開発者や顧客は、この4つの変数にいつも悩まされている。何故なら、常に相反する項目として私たちの前に立ちはだかるからだ。



求めてしまうもの

まずは、コストと時間を見てみよう。誰もが早く安価にシステムが出来れば問題はないだろう。しかし、ここで注意する点は、システムのライフサイクルから見ることである。当初のコストは安くてもシステム開発が進んでいけばいくほど、変更が発生した場合のコストは高くつくことになる。ましてや、システムが稼働後の変更は当初の開発コストより多くなる。そして、それは時間が経てば経つほどコストは上昇する。もし、これらの変更コストを比較的安価で常に変わらないものに出ることに出来れば、誰もが悩まなくて済むであろう。

次に、品質とは何かである。提供する機能が優れていて、かつ、不具合等のバグが発生しないことと考えるのではないだろうか。これは残念ながら完璧には出来ないであろう。そして、限りなく完璧を求めるとコストと時間は無限大となり、いつまで経っても顧客にシステムはリリース出来ないのである。何かおかしい。

実は、スコープこそがそれを解く重要な変数である。スコープを訳すると「有効な範囲」となる。この「有効な」が大事なのである。それは、顧客がそれを提供された時点で必要としている機能を満足させ、その機能を支障ない範囲で使えることが良いということである。とすれば、品質はその範囲を満足させればよいことになる。

しかし、注意してほしい。「それが提供された時点」ということである。スコープは時間とともに変化するのである。一つの機能が提供されたことにより、次の機能の要求は変わるし、企業

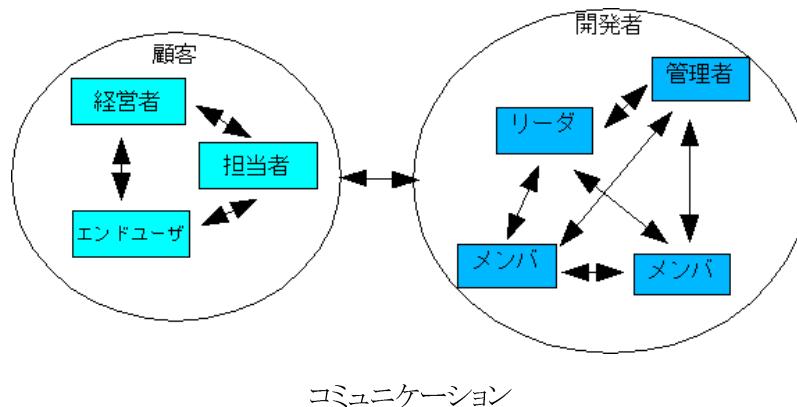
の内部環境の変化、企業にとっては外部環境となるその企業の業界、世の中の政治、経済、社会の環境変化によって変わるのである。

そう、時間の変数はスコープと密接に関わっている。短い時間で適当なスコープを繰り返し提供していくことが本当は大事なのである。

### 3. 2. 4つの価値

#### 3. 2. 1. コミュニケーション

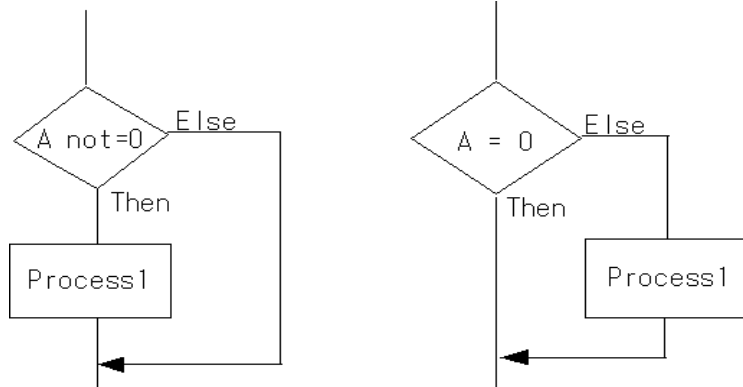
システム開発は一人では行えない。最低、顧客1人が必要である。ちょっとしたシステム開発規模になればプロジェクトで行う。そして、多くの人とそのプロジェクトに入ってくる。このとき、システム開発を成功に導くにはコミュニケーションが重要である。皆さんは、こう聞くと「当たり前じゃない。」と吐き捨てるように言うであろう。だからこそ、重要であり、価値があるのだ。プロジェクトがうまくいかなかったときのことを考えてほしい。どこかでコミュニケーションがとられていないはずだ。



#### 3. 2. 2. シンプル

シンプルという言葉からは、“Simple is Best.”という、一昔前のコマーシャルの言葉を思い出す。シンプルを日本語にすれば、「単純」「簡単」ということである。

ということは、伝え易いということにもなる。「……すればわかる」とよく言うが、シンプルだからこそ、「……すればわかる」と言えるのではないだろうか。



どちらも同じ

図を見てほしい。どちらも実現することは変わらない。シンプルな設計・シンプルなコーディングなら右になる。これは簡単な例だから左でもかまわないが、これが複雑になるとどうなるかである。一見、右は複雑になりそうだが、人が思考するという観点からみれば右は単純なのである。

### 3. 2. 3. フィードバック

フィードバックは、システム開発のすべての事柄に当てはまる。そして、開発中には、なかなか出来ないものである。一つの変更があればそれに関わるであろうすべての項目のテストをし直すとか、ひとつのプログラムを結合するときにはそれに関わるであろうすべての項目のテストをするとかである。

顧客が新たな要求を提示したときも、計画からのフィードバックをし、全体の進捗のスピードやモチベーションの変化を考えているだろうか。

フィードバックはあとで起こる災いで苦勞するより、今その災いに対応する方がずっと楽であるからだ。

そして、コミュニケーションが図られて、シンプルであればフィードバックはし易く、フィードバックすることで、コミュニケーションが良くなり、よりシンプルになるのだ。

4つの価値は相互に深く関連しているのだ。そう、これをするためには「勇気」が必要なのだ。

### 3. 2. 4. 勇気



XP において私は、「勇気とは、いろいろな場面で短時間に判断する意思と行動」のことではないかと考えている。システム開発において判断しなければならない場面は多い。その場面でタイムリーに判断しないと良い結果には到達しない。確かに、誤った判断は悲劇を招くであろう。しかし、判断しないで失敗するよりは……。

4 つの価値について語ってきたが、皆さんは、「プロジェクトマネジメントと何が違うのか」と思っているであろう。そう、プロジェクトマネジメントにも通じる価値なのである。この価値を皆さんのプロジェクト管理に取り入れてください。そこから、XP 導入の第一歩としましょう。低いハードルから飛び越えて行けば良いのです。そうすれば、次の高いハードルに対して踏み台を用意することになるのだから。





## 第4章 基本原則 ～ 本当に当たり前と言えますか ～

### 4. 1. 瞬時のフィードバック



フィードバックするには、勇気が必要である。ましてや、瞬時のフィードバックとなるとかなりの勇気(判断力と言った方がよいかもしれない)が求められる。

XPのサイクルは、確かに、PLAN-DO-CHECK-ACTIONサイクルに似ている。このPDCAサイクルで重要なのは CHECK-ACTION である。だから、フィードバックが重要なのは理解できるであろう。

しかも瞬時のフィードバックである。確かに、誤ったものをフィードバックするのは、すぐに戻す方がコストはかからない。

私の愚かな頭の中で考えを進めていくと、ますます、瞬時のフィードバックが大きな意味を持つてくる。簡単な言葉だが、その要求は非常に厳しいものである。

### 4. 2. シンプルの採用

シンプルと言えば、昔のテレビコマーシャルで「Simple is Best」という言葉があった。何か、心に残った言葉である。

何かシステムを作る場合、簡単なシステムや単純なシステムは誰でも理解することは容易であるが、複雑なシステムは容易に理解することはできない。そのため、シンプルということが大切であることはすぐにわかる。

しかし、それを実際にシステムに作ろうとすると、シンプルなシステムほど作りにくいシステムはないだろう。すぐに複雑なシステムになってしまう。その方が、とりあえず手は出しやすいのである。が、しかし、いったんそのシステムが出来てしまうと、もはや、誰も手を出せない複雑なシステムになってしまっているのである。

一見、単純なシステムは簡単そうに見えるが、実は、そうではなく良く熟慮されて作られているのである。だから、誰でもが理解し、手を加えられるのである。

シンプルという言葉の響きは良いが、現実には、重たい言葉である。

#### 4. 3. インクリメンタルな変更



一度に大きなシステムをリリースする開発プロジェクトや、システムへの大きな変更を一度にリリースすることには大きなリスクを伴うし、一歩まちがうと、小さなずれが大きなずれとなり、取り返しのつかない状況になる。そのためには、小さな単位で、継続的にリリースをして進めていく形が良いのではないか。

小さなずれは小さなずれで吸収が容易であるし、何よりも明確に終わったもの(リリースしたもの)がわかるというのは、開発者にとっても顧客にとっても全体の進捗が把握しやすいものである。

小さなずれは小さなずれで吸収が容易であるし、何よりも明確に終わったもの(リリースしたもの)がわかるというのは、開発者にとっても顧客にとっても全体の進捗が把握しやすいものである。

実は、このことはリスクマネジメントがきちんと考えられているということなのである。インクリメンタル変更を、表面上の言葉だけで捉えて安易に採用してはいけない。リスクマネジメントとしてきちんと成り立っている形でのインクリメンタルな変更が求められているのである。

#### 4. 4. 変化を取り入れる

XP に批判的な人には、この原則は理解されにくい。というよりは、そういう人々はこの原則を理解したとらない。

実は、システムというものは、当初考えた仕様で満足のゆくことは、小規模なシステムを除けばそう多くはないのである。しかも、そのシステムのライフサイクルから見れば、仕様変更がないシステムはほとんどないであろう。

その事実がある上に、昨今の変転の激しいビジネスでは、要求仕様を決定した時点でパーフェクトであっても、リリースする時期にはそうではなくなっているという状況になってしまっているのである。

では、どうすれば良いのか。短期間でリリースできる要求単位で開発し、変化する要求に対応して行けば良いのである。そうすれば、常に、一定のコストで顧客を満足させつづけることが可能となるのである。

これは、大きな発想の転換である。しかし、より現実を注視したもの、もしくは、現実に沿った開発なのである。

これを成功させるには、顧客も開発者も開発者を管理する人もすべてが、ビジネスの変化を認識し、開発スタイルもそれに対応したものに变化させなくてはならないという認識に到達する必要がある。経営レベルの視点が求められるのである。

#### 4. 5. 質の高い作業

開発者は、誰もが良いものを作りたいのである。本当に良いものを、である。形式的な品質ではなく、そのシステムが利用者に本当に要求されているものを作りたいのである。

そのためには、そのような質の高い作業ができる環境が重要である。形式的な品質のために余分な作業やコストが発生し、開発者の意欲を低減させては何にもならない。

質の高い作業は、コスト高になるとは限らない。本当に質の高い環境を用意すれば、そのシステムのライフサイクルにかかる費用は安くなると言える。

しかし、質の高い作業を提供できる顧客、いや、開発者の管理者がどれほどいるであろうか。これは、XP の話だけではなく、すべてのシステム開発作業を質の高いものにするために重要なことである。





## 第5章 12 のプラクティス ～ 深海の底に辿り着けるか ～

ここでは、12 のプラクティスを説明するわけではない。XP が軽薄な技術者にもてはやされ、訳のわからない人達に大きな誤解を与えるのではないかという危惧から、少し違った観点から述べることにする。

そう、少し気になる5つのプラクティスについて述べよう。

### 5. 1. 短期リリース ～変化するという事実～



#### 5. 1. 1. 顧客サイドが重要

短期リリースをする重要性は、顧客サイドにもある。この点をしっかり理解して欲しい。

まず、システムの開発期間が長ければ、顧客をとりまく外部環境や顧客自身の内部環境が変化しているということである。特に、近年におけるビジネス環境の変化は速く、大きく変化する。そのような状況で、当初の顧客の要望が、リリース時点で同じであるとは言えない。これは、顧客と開発者両者の責任である。当然、開発されたシステムはリリース時点から仕様変更が発生し、そのコストで顧客と開発者がもめるものになる。

では、どうするか。簡単である。変化しない程度の期間でリリースすることである。そして、ここで重要なのはその短期間で顧客が運用できる範囲内しかリリースしないことである。すべてを短期間で開発することは不可能であるが、それらのすべてを運用できる形にもっていくことも、顧客側には不可能なのである。

#### 5. 1. 2. 顧客は変化する

ここまでは、なんとなく納得された方も多いと思われるが、実は次のことを本当に理解しないとイケないのである。

顧客はリリースをされたシステムを運用することにより、変化するというものだ。もし、リリース前とリリース数ヶ月後で顧客が変化していなければ、システムを導入しなくても良いということになる。しかも、やっかいなことにはどう変化するかわからないのである。

当然、顧客の要求は変化してくる。このことは、顧客サイドが要求管理をしっかりと行う必要があるということである。

短期リリースを継続的に行うということは、顧客自身も地に足のついた対応が必要だということだ。

最後に、蛇足ではあるが、開発サイドのリーダーやマネージャーは顧客が変化することを理解していないなら、プロジェクトを引き受けてはいけない。

## 5. 2. テスティング ～発想の転換～



XP において話題を集める項目のひとつなのがテストングである。特に、テストファーストは鮮烈である。

だが、巷にあふれるのは、まずテストプログラムのコーディングやJUnit などの xUnit を利用することだけが先行する言説だ。時として、これが XP だと唱える意味不明な宣教師がいる。

### 5. 2. 1. テストは Requirement を保証する

顧客の Requirement を保証するにはテストングすることが重要である。それは、ユニットテスト、システムテストや受入テストのどの時点でもそうである。もっと極論すれば、顧客の Requirement を保証するテストで正当な結果を出せるプログラムが本当に正しいプログラムと言えよう。

あまりにも当たり前のことである。そして、顧客の Requirement を保証するということは対象となるすべてのプログラムが満足しなければいけないことである。

そろそろ本論を述べよう。

### 5. 2. 2. Requirement を保証するためのテスト項目を列挙する

まず、しなければならないことは顧客の Requirement を保証するためのテスト項目を列挙することだ。それも単に挙げるのではなく、顧客の Requirement を保証する結果をも考える必要がある。そこから、それを保証するのに必要なものを用意する必要がある。出来れば、既にあるものを使う方が信頼性は高いのは当然である。だから、JUnit のようなテストプログラムを利用するのである。何も、テストプログラムがすべて必要であるとは言っていない。その中身を表示するだけで OK なら、プリントユーティリティを起動させても良い。それで、十分に、早く、簡単に確認できるならばそれで良いのである。無理して、xUnit を使わなくても良い。

次に、出来れば、既にあるプログラムも同時にテストしてくれれば、便利である。これも、だからと言って、複雑なテストプログラムや膨大なテストプログラムになるのなら、そのシステムの開発規模やコストとを考慮して実装するかどうかを判断して欲しい。

### 5. 2. 3. 発想の転換

テスト項目とその結果があれば、それに対応するテストプログラムは明確に作成できる。そして、提供するプログラム自身も明確に実装できるのではないだろうか。提供するプログラムの機能から考えて実装しても、顧客の Requirement に添うものになるだろうか。むしろ、顧客の Requirement を保証するテスト項目と結果から実装する方が、顧客の満足度は高いのではないか。

XP がテストファーストを重視するのは、まさに、顧客の Requirement を保証することそのものだからではなかろうか。従来の方法のように機能を考え、そしてテスト項目を考えていたのでは、顧客の Requirement を保証するには大変である。テストを最初に持ってくることは、顧客の Requirement を保証するプログラムしか作らないということなのである。

ウォーター・フォールの開発手法であっても、このテストファーストは取り入れるべきである。なぜなら、顧客の Requirement を保証することが一番重要だからである。



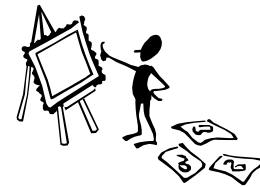
### 5. 3. ペアプログラミング ～技の激突、そして融合～



XP においては、テストिंगとともにペアプログラミングがその特徴として語られる。しかし、私はそのメリットを通常とは少し違う観点で見ている。

#### 5. 3. 1. 生産性と品質

ペアプログラミングは、達人(もしくは匠)のペアであれば個々で行うよりも、生産性は高く、品質も高くなるのである。それは、達人は頭の中で既にそのプログラムが出来ているからである。それをお互いがぶつけることにより、瞬時により良い形が導き出される。



しかし、普通は達人のペアでない場合が多い。この場合は、従来で言うところのコーディングベースでの生産性は1人で行うより落ちる。ただ不思議なことに半分にはならないのである。落ちるのは、20%～30%程度である。これは、1人で行うと、考えてしまうことがあるのに対して、ペアで行うと、もう1人がアイデアを出すことにより考えている時間が短くなるからである。このままでは遅れてしまうのだが、システムをリリースするまでだと生産性は変わらなくなる。しかも、リリース時点の品質はペアプログラミングの方が高くなる。これは、1人の目で見ると、2人の目で見ると正しく検証できるという当然の理由からである。

#### 5. 3. 2. リスクマネジメント

それだけなら、ペアプログラミングはそんなに価値がないように思われるが、非常に大切な点からメリットがある。リスクマネジメントの観点からである。



一つは、よく言われていることだが、担当者が事故や病気によっていなくなっても、開発スピードは落ちるが、誰もそのプログラムを知らないということがないことである。これは、プロジェクトマネージャーとして当然考慮しておかなければならないリスクマネジメントのひとつである。事故や病気だけでなく、退職や他のプロジェクトに引き抜かれる(というよりは、呼び戻されると言うべきか)ことにより、1人がそのプロジェクトからいなくなることは当然考えなければいけないが、同時に2人ともそうなる確率は低い。常にペアを組み合わせることは、まさにリスク分散である。そして、ペアをプロジェクト内で組替えて行くことにより、すべてのプロジェクトメンバーがそのプログ

ラムを知っているという共有化への第一歩を踏み出すことができる。

もう一つは、システムリリース後に起こりうる障害の可能性を低くすることである。複数の人が1本のプログラムを見ることは、潜在的に隠れている障害を事前に取り除くことを意味している。1つの障害がなくなることは、リリース後に発生する余計なコストを無くすことである。本来の生産性を考えるのなら、発生してはいけないコストを産み出さない施策を考えておかなければならない。1人しかそのプログラムを知らないということであれば、その担当者がそのとき対応できる可能性は少なくなってしまう。別の人が対応するなら、多くのコストを費やすことになるしかない。

システムのライフサイクルから考えれば、ペアプログラミングの生産性と品質は非常に高いものになる。これは、起こりうるであろうことに対する予防策と同時に対処策も考えているからである。とはいえ、リリース後も、プログラムを変更するときはペアプログラミングできる方が良い。リリース期間に対して、求められる品質は高いはずである。それを保証するためにもテストファーストとペアプログラミングは重要である。

### 5. 3. 3. 技の激突、達人への道

ペアプログラミングは、同等スキルの人でペアするべきであると私は考えている。新人のOJTや、下級エンジニアと上級エンジニアを組み合わせ、その中で教育しようと考えているならペアプログラミングは止めたほうが良い。何故なら、今まで話したメリットはなくなってしまうからだ。そして、私がペアプログラミングのメリットとして考える一番大事な項目が達成できないからだ。



ペアプログラミングは、技と技との真剣勝負である。しかも、一つのゴールに向かってプログラミングしていかなければならない。ということは、相手を徹底的に打ちのめすことではなく、創造していかなければならないのだ。技と技とを激突させ、切磋琢磨し、一つのを創造していくことは、最終的には、技と技との融合を導き出し、より良いものを創造させる。そして、技術者はスキルをアップして行くのである。決して、仲良しクラブではないのである。

ペアプログラミングに隠されたメリットとは、技術者が達人の道を歩めるということである。

そして、達人になれば、このペアプログラミングの最初に話した、生産性も品質も高いものを提供できるのである。もはやこの段階にすれば、人月の工数ではなく、達人たちの価値や達人が創造したシステムの価値で費用を払わなければ、彼らにプログラムは作ってもらえないであ

ろう。しかもやっかいなことに、達人が創造したシステムの品質には、普通の技術者は到底到達しえないものなのだ。

#### 5. 4. 40 時間労働 ～本当の生産性～

40 時間労働は、XP を推進するプログラマには福音のように響く言葉である。しかし、本当はこれほど過酷な事はない。

これは、常に、MAX の生産性を確保しつづけることを言っている。

最大の生産性をキープできる時間数で行えば、一番コストが安くなるという当たり前のことなのだ。



しかし、残業をせずに時間がきたら仕事を中断してしまうことを言っているのではない。

当然、きりの良いところで仕事は終らせた方が生産性は良い。だからと言って、毎日遅くまで仕事をしていけば生産性を劣化する。コストで見ればマイナスである。一番生産性が高いレベルでスケジュールする方が良いということだ。

こう考えると、40 時間労働は非常に恐いプラクティスである。

人が人として働ける最大限の時間で最も生産性が高くなる時間数を設定することを、ここでは言っているのである。

プロジェクトマネージャーは、メンバーを人として考え、モチベーションを高く持たせながら、コストと生産性が最大限になるところの時間数を設定して欲しい。



## 5. 5. コーディング規約 ～見えないドキュメント～

ここでは、コーディング規約について述べるのではない。XP ではプロジェクト内に入ると、コーディング規約こそが文書として見えるであろう。



### 5. 5. 1. 内部のドキュメントがない訳

ストーリーカードやタスクカードなどのドキュメントもあるが、文書化という点ではコーディング規約だけである。

実は、これが、XP 論者が短絡的にドキュメントは不要だと勘違いしているところであろう。

欧米は、契約社会である。実は、そのプロジェクトをはじめるときやメンバーになるときには、なんらかの契約が取り交わされているのである。ときとしてそれは、日本においては分厚すぎるほどのドキュメントである。

そして何よりも、大事なことは、個々のメンバーが行うことや役割が明確であり、そのための権限の委譲と責任が明確になっていることである。

言わば、外に対しては初めにきちんとしているのである。だから、内部ではドキュメントは不要なのである。

### 5. 5. 2. 本来、ドキュメントを作成する人

もう一つ、本来作成すべき人がドキュメントを作成していないことや、A4 用紙 1 ページで表現する技術があればそれで良いということが理解できない人達が多い点を指摘しておきたい。

XP はオンサイトカスタムである。プロジェクト内にいる顧客が、進捗報告を顧客の経営者等に報告する必要がある。このため、日本においては、開発サイドは顧客が進捗報告を作成できるスキルを伝授する必要がある。顧客は自分のシステムを自分自身で作成しているのだから、顧客自身が進捗報告を作成できないといけないし、開発者の進捗も顧客の視点で報告されなければ、本当の問題点は見えてこない。

そして議事録はプロジェクトマネージャーかプロジェクトリーダーが直接書くものである。

内部に対しては、A4用紙1ページなら、ホワイトボードに貼り付けければ良い。外部に対してならば、プロジェクトマネージャーやプロジェクトリーダーは自分自身のプロジェクトマネジメントを問われていることであるから、自分自身で書かないでどうマネジメントするというのか。最低限、議事録は、自分自身で書くべきである。

あとは、必要なときには、原則1ページでドキュメントを作成するようにした方が良い。

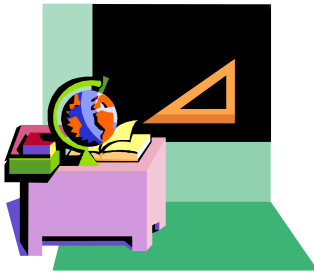
ここで重要なのは、役割・権限の委譲、責任を明確化しておくことである。ドキュメントの作成を指示した人も、作成した人も含んで明確化することを忘れずに。



## 第6章 明日を担うもの ～ 哲学と両輪の輪 ～



### 6. 1. XP は哲学か



サイトに掲載をはじめた時に、「技術論的なことがなく、哲学みたいですね。」と言われた。

そのとき、「哲学でしょうね。」と答えた。

これは、新しい開発手法が進めるには、まず、ビジョンやポリシーが必要となるからだ。ビジョンやポリシーを述べるときには、どうしても哲学になってしまうのではないだろうか。何か新しいものを提唱するには、技術論だけではなく、哲学の世界も必要なのだ。

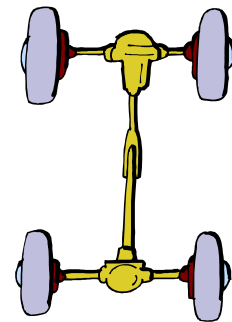
これは、再帰しているのではなく、次の段階を語るには哲学的なものからスタートする必要性があるということだろう。

私の今までの戯言も哲学的なことしかないかもしれない。技術論はゼロと言った方が良い。

### 6. 2. ソフトウェアプロセスとの関係

最後に、ひとつ述べておきたいことがある。それは、ソフトウェアプロセスとの関係である。

ソフトウェアプロセス改善で SPA や CMM がブーム気味に騒がれている。XP は目の敵にされたり、マッピングされたりしている。



XP は開発手法である。ソフトウェアプロセス手法ではない。このことを理解している人が意外と少ない。理解していれば、批判したり、マッピングしたりはしない。要するにそれを唱えている人の成熟度は、本当は低いのかもしれない。

情報システムを開発するには、ソフトウェアプロセスと開発手法がバランスのとれた両輪の輪でなければならないということだ。ウォーター・フォール型テンプレートのソフトウェアプロセスで XP を見てはいけない。XP にあったテンプレートでソフトウェアプロセスを行わなければならない

い。

私の個人的な考えであるが、XP をうまく行えるプロジェクトは、ソフトウェア成熟度が高いことを前提としている。XP では、モニタリングや最適化を常に継続的にできることが求められる。短期リリースをすることから、そのサイクルは非常に短期間で継続的に速いスピードで回って行く。しっかりした成熟度を持ったプロジェクトでないとその流れに吞まれてしまい、うまく開発を進めることは出来ない。

しかし、XP を数千人月のシステム開発プロジェクトに適用することは避けるべきである。そのようなシステムは重厚長大なシステム開発手法がマッチする。これは、組織の中の組織を管理していかなければならないからだ。組織の中の人を管理できる範囲なら、XP は可能であろう。(例えば、学校を考えると、クラス程度なら人として管理、学年となると組織として管理となるであろう。)

いずれにしても、ソフトウェアプロセスと開発手法は、ともに助け合うもので相反するものではあってはならない。うまくバランスを取り、両輪の輪として適用して欲しい。

### 6. 3. 明日を担うもの



今、欧米において、ソフトウェア開発手法は新しいステージに移ったのではないかと感じる。なぜなら、XP を含めた Agile 開発手法について、一過性のブームではなく、多くのオブジェクト指向技術の著名な人々が語り始めているからだ。

これは、XP を含めた Agile 開発手法が、明日を担うソフトウェア開発手法なのかもしれないからだ。

だが、日本においては、「ソフトウェアの失われた 10 年」により、オブジェクト指向技術が一つの開発手法として確立されたとは言い難い。ましてや、この新しい開発手法のステージについてとなると哲学的なこととなり、ますます理解されにくくなっている。

そのような状況では、XP がただのブームとして、Agile が風疹のように騒がれるだけに終わることは十分危惧される。

Kent Beck の愛読書は、宮本武蔵の五輪書のようなものである。ひよっとしたら、XP の極意は五輪

書かもしれない。そうだとしたら悲しいことである。日本の心が日本で受け入れられないことになるからである。

XP が明日を担うもの、もしくはそれを導くものならば、この戯言も何かの一石を投じたのではと考える。非常に、小さな石ではあるが。

しかし、明日を担うものを望まない人達には、戯言もただの塵と化すであろう。

今日、そして明日においてもソフトウェア開発者を名のり続けたい人には、明日を担う開発手法を求めて行って欲しい。望まない人には、必要ないであろう。彼らには明日がいないのだから。



## 付録 参考文献

1. Martin Fowler, <http://martinfowler.com/> ファウラーのXPについてのドキュメントが掲載されている。
2. Kent Beck, *Extreme Programming Explained:Embrace Change*, Addison Wesley Publishing Company, 1999. ([邦訳]長瀬嘉秀, 永田 渉, 飯塚麻理香(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』, ピアソン・エデュケーション, 2000.)
3. Pete McBreen, *Software Craftsmanship : The New Imperative*, Addison Wesley Publishing Company, 2002. ([邦訳] 村上雅章(訳),『ソフトウェア職人氣質』, ピアソン・エデュケーション, 2002.)
4. Peter F. Drucker, *Management Challenges for the 21st Century*, Harpercollins 1999. ([邦訳]上田惇生(訳),『明日を支配するもの—21 世紀のマネジメント革命』, ダイヤモンド社, 1999. )



#### 著者紹介

新保 康夫(しんぼ やすを)

ITコーディネータ・ITコーディネータインストラクタ

日本 XP ユーザグループ関西支部 関西代表(<http://www.xpjug.jp/>)

ITコーディネータ・パートナーズ 理事(<http://www.itcp.jp/>)

[shimbo@osa.ncs.co.jp](mailto:shimbo@osa.ncs.co.jp)



---

発行日 平成 14 年 11 月 13 日 初版 発行

著 者 新保康夫

発行者 日本コンピューター・システム株式会社

(<http://www.ncs.co.jp>)

〒530-0005

大阪府大阪市北区中之島 3-2-18 住友中之島ビル

Copyright © 2002 by SHIMBO.

Printed by Nippon Computer System Co., Ltd.





<http://www.ncs.co.jp/>