解　説

ベスト・プラクティスの思想と実践

Concept and Implementation of Best Practice by Tomoo MATSUBARA(Matsubara Consulting. IEEE Software).

松原　友夫[1]

1 松原コンサルティング／IEEEソフトウェア

1. はじめに—背景

ソフトウェアが工業製品であるという側面から，ソフトウェア組織の多くはハードウェアで成功をおさめたプロダクト中心の品質管理を長い間継承してきた．たしかに，そのやり方のいくつか，たとえば計測値による管理は，初期のソフトウェア開発の改善に貢献した．しかし，ソフトウェア・プロジェクトは，しばしば大幅な納期遅延や予算超過，そして稼働後には重大な障害を引き起こし，システムの規模と複雑性の増大にともないトラブルも増加した．人の頭脳集約作業の成果として創られ，ディジタルで無形物という特性を持ち，しかも進化するソフトウェアの改善は，プロダクト中心のアプローチではうまくいかないことが次第に分かってきた．代わって，ソフトウェアではそれを創るプロセスに注目すべきであるという共通認識が生まれ，それまでの中間成果物を含むプロダクト中心の改善活動は，プロセス中心へと移った．

産業をあげてのソフトウェアのプロセス改善は，2つの別の流れから始まった．その1つは，対象を限定しない製品やサービスの品質を維持するための要件として，1987年に制定されたISO 9000品質システム規格のソフトウェアへの適用であった．これは，主として伝統的な製品，つまりハードウェアで適用されてきた品質管理の思想に基づいて作られていたため，無形物であるソフトウェアを対象とする場合は，設計の部分のみが該当し，規格条文はソフトウェア向けに解釈し直す必要があった．もう1つの流れは，米国カーネギーメロン大学に属するソフトウェア工学研究所(SEI)の実施能力成熟度モデル(CMM)である[1)]．これは，SEIが米国国防総省の委嘱をうけて1991年に初版を出版した，組織のソフトウェア・プロセス成熟度を5段階で評価するモデルである．これの利点は，ISO 9000とは異なり，ソフトウェア固有のプロセスを対象としていることと，それが，段階的な成熟を前提として作られていることである．

プロセス改善の目的は，本来適用することによる品質と生産性の効果にあるのだが，ISO 9000とCMMが公式の認証や評価という組織にレッテルを貼る制度と結びついたために，改善より認証取得や評価レベルが注目されるようになった．特に，日本ではISO 9000の認証制度により高い関心があるが，改善の目的が本来の品質改善から認証の取得にすり替えられ，地道な品質/生産性改善努力を妨げる，という歪んだ現象を，一部にもたらしている．

一方，CMMは，米国国防総省を始めとする政府機関が，ある成熟度レベルの達成をソフトウェア開発プロジェクトに応札する条件としているところから，米国の特に防衛関連の産業がこれで評価を受けることに積極的である．いままでは，CMMによる評価はほとんど米国内に閉じていたが，最近はSEIもCMMの国際化を意図しており，基本文書の翻訳や国際的な評価も活動の視野に入れている．すでに，CMM文書の日本語化の話も進んでいる．

さて，これら2つのモデルの存在は，産業にとってはなはだ不便である．特に，ヨーロッパと米国のビジネスに関わる国や組織では，これら2つのモデルで認証/評価を受ける必要がある．ソフトウェア工学関連の標準化を担当する国際標準化機関，ISO/IEC JTC1/SC7は，前述の2つの流れの統合を求める国際的な要望に答えて，1992年にソフトウェア・プロセス評価を新作業項目とすることを承認した．このプロジェクトは，現在規格の前段階としての技術報告を作成すべく作業中である．

CMMは，ソフトウェアの特性に注目したプロセス改善としては一歩前進であるが，ISO 9000と共に，これが組織のプロセス改善を指向していることから，トップダウンで公式的に，画一的に実施される傾向がある．主としてこうした観点から，CMMでさえ，特に米国に多いソフトウェア製品会社と，そこで働く独立独歩のスーパープログラマからの強い反発がある．反対論の急先鋒であるJames Bachは，次のように述べている[2)]．

(前略)CMMは，よく見たとしても，組織進化の単純なモデルに沿ってグループ化された，特定のソフトウェア工学の理論家や実践家の間の最良のコンセンサス

にすぎない．悪く見ると，CMMは，ソフトウェア開発の真の動的な性格を覆い隠し，それに代わるモデルを作ることを抑圧するものだ．（中略）こうした理由から，CMMは，ソフトウェア製品を開発している高度に競合力があり革新的な多くの会社では，ほとんど知られていない．

さらにBachは，CMMの基本的な問題点として，次の7点を指摘している．

- 果たして狂気の世界で実践し得るか？
- それが基礎とする理論がない
- 実験で実証されるべき根拠がはっきり示されていない
- プロセスを重視して人を無視している
- プロセスの制度化に重点を置いている
- プロセスの動的な性格をほとんど考慮していない
- 目的のすりかえを奨励する

つまり，彼は，画一的なプロセスの公式的な実施は，状況が多様で変化の激しいソフトウェアの世界では適用できないことを指摘している．

2. ベスト・プラクティスとは何か

ベスト・プラクティスは，ISO 9000，CMMのような，公式的プロセス改善モデルのトップダウン的な実施に代わる，より現実的なアプローチとして，最近米国で提唱されたボトムアップ指向のプロセス改善活動である．もちろん，トップダウンとボトムアップの間に厳然とした違いがあるわけではなく，両者が混在するのが普通であるが，改善のトリガがボトムにあり改善の具体策が，良い成果を生んでいる開発現場からの発掘からもたらされる，という点で，ボトムアップなのである．

ボトムアップ指向からくるもう1つの特長は，多様性である．つまり，それぞれの組織がおかれた多様な状況のもとで，もっとも効果があったプロセスを広めるため，開発している製品の種別，開発環境，技術者や管理者のレベル，組織の伝統や文化，といった状況が多様であれば，当然ベスト・プラクティスも多様でユニークなものとなる．それは，しばしば社内方言を用いて表現される．ほかで行われているベスト・プラクティスを借用することも初期にはありうるが，いずれは自らのものを作りあげていくことが望ましいとされる．

3. ベスト・プラクティスはなぜ生まれたか

プロセス・モデルを用いたプロセス改善は，公式性が重視される．情報の世界は，技術や環境の変化が激しい狂気の世界であり，変化に対処するために，環境の変化に応じてプロセスをダイナミックに変えていかねばならないが，プロセスが基準化され，それへの適合性が監査者から評価されるから，プロセスは硬直したものになりがちである（CMM擁護派は，当然この主張に同意せず，これは大きな争点になっている）．

また，ビットレベルの細部まで厳密性が要求される頭脳集約的なソフトウェア開発では，開発の当事者以外がプロセスの細部を理解し，適切な改善策を指示することはほとんど不可能に近い．

さらに，プロセスの質は，それを遂行する人のモラールに大きく左右されるが，公式モデルから出発する改善は，それを実施する技術者が，やらされるという受け身の立場に立たされることが多く，なかなか成果が上がらない．

こうした硬直性，受け身，そして現場から遊離した改善策，といった欠陥に対処するものとして生まれたプロセス改善のアプローチが，ベスト・プラクティスである．またこれは，現在組織のなかで行われている，その組織にとって最善のプロセスに基礎を置くため，プロセス改善の実効を上げやすい．

4. ベスト・プラクティスの実例

ベスト・プラクティスは発掘するものであることはさきに述べたが，これらが実際にどんなものであるかを知るために，また最初の手がかりとして利用するために，すでに出版され公表されているプラクティスをいくつか紹介しよう．

4.1 国防総省のベスト・プラクティス

米国国防総省は，1994年頃ベスト・プラクティス推進活動の一環として，それを審議する会議を開催した．それが定例的に開かれた場所の名をとって，それはAirlieソフトウェア審議会と呼ばれた．審議会の参加者は，品質の権威者のV. Basili，ピープルウェアで著名なT. DeMarco，メトリックスの権威者C. Jones，著述者でコンサルタントのE. Yourdonなど，14名の指導者であった．そのメンバからの170件の提案を要約したのが，次の9件のベスト・プラクティスである[3]．要約であるためか，やや公式的なものになっている．

(1)公式のリスク管理

これには，最低次のことが含まれる．

- リスク管理担当者の指名
- リスクが顕在化した場合の日程，費用，その他の予備バッファを計画に盛り込む
- リスク・データベースの作成
- リスクの兆候を識別するためのリスク・プロファイルの作成

• リスク計画の継続的な更新と観察

(2)仕様書としてのユーザ・マニュアル

建て前としては，開発着手以前に仕様が確定しているべきであるが，それが単なる理想にすぎないことを，開発現場を知る人はよくわかっている．そこで，審議会は，現実的なアプローチとして，設計とインプリメンテーションに入る前に，最初にユーザ・マニュアルを作成し，これを仕様書の統合的な部分とすることを提唱している．

(3)審査，レビュー，およびウォークスルー

特に説明を要しないが，これが適切に実行されていないことから，ここに提案されたのだろう．

(4)メトリックスに基づいたスケジュール管理と追跡

これも説明を要しない．審議会は，予測ツールの利用を推奨している．

(5)小さなマイルストーンでの二進品質ゲイト

審議会は，進捗報告でよく見られる，いわゆる90%シンドロームを避けるために，小さなマイルストーンごとに，客観的なバイナリ合否判定基準を設けることを推奨している．これは，後で述べるMicrosoftのベスト・プラクティスでは，毎日のシステム構築というかたちで実施されている．

(6)プロジェクト計画における全体の可視性および進捗と計画の対比

これは，プロジェクト全体の進捗を視覚的に見るプロジェクト・ダッシュボードの設置と，実際と計画の対比を意味するが，これにはプロジェクトの進捗を常に隠さずに示す文化も含まれる．

(7)品質目標に対する欠陥の追跡

審議会は，「欠陥は，プロジェクトの各過程で公式に追跡されねばならない．このアプローチでは，記録しないで摘出される私的な欠陥はあってはならない．残存する，あるいは潜在する欠陥の計算と同様に，最初の品質目標は，テスティング作業の期間中，実際の欠陥数と比較されねばならない」と述べている．

(8)ハードウェアの仕様とソフトウェアの機能性の分離(説明省略)

(9)人員編成は経営者の責任

審議会は，「経営者は，適任の技術者によって，プロジェクトを編成する責任がある」と述べている．これは当然のことだが，多くの場合，果たされないことから，これがあげられている．

4.2 Alan Davisのベスト・プラクティス

コロラド大学教授でIEEE Software誌の編集長であるAlan Davisは，ベスト・プラクティスのもとになるものとして，ソフトウェア開発に共通する一般原理を「ソフトウェア開発201の鉄則」[4]にまとめた．そのなかから一部を抜粋しよう．ここに示されている原理は，かなり具体的である．

• 原理43：なぜこの要求項目が含まれたかを記録せよ

なぜこの項目が含まれたのかがわからなければどう変えてよいかもわからない．

• 原理50：要求に優先順位をつけよ

すべての要求項目は平等でない．優先順位が識別されていないと，努力と顧客の満足度をバランスさせることができない．

• 原理52：どの要求項目にも識別番号をつけよ

品質確保にもっとも重要である追跡可能性は，単に記録を残すだけでは達成できない．要求項目からテスト結果に至る，連綿とした項目単位の追跡の基礎は，各要求項目に識別番号をつけることから始まる．Davisは，これを機械で自動的に行う方法も提案している．

• 原理62：設計から要求項目を追跡せよ

原理52の実施によって，これが可能になる．

• 原理74：変更が容易なように設計せよ

設計とは要求仕様を単に実現することだけではない．変更が必ずあることを念頭に入れて，変更が容易な構造を作るのも設計の一部である．

• 原理83：アプリケーションを熟知せよ

最適な基本構造やアルゴリズムは，アプリケーションのユニークな特性をよく知っていなければ作れない．

• 原理88：グローバル変数を使うな

だれかがいじるかもしれない変数は，摘出困難なバグの原因になることが多い．

• 原理91：意味のある名称を使え

意味のない名称はコメント行を増やし，かつ理解し難いプログラムを生む．

• 原理95：コードを仕上げる前にコメントを加えよ

これは自明だが，いまだにコメントのないコードが横行している．

• 原理107：テスト項目を要求項目と関連づけよ

原理52を基礎として，要求項目の関連づけを，設計(原理62)だけでなくテスティングでも行う．原理50が実施されていれば，これによってテストの優先順位もつけられる．

• 原理109：自分のソフトウェアを自分でテストするな

初期のテストは自分でやってもよいが，ほぼ仕上がってからのテストを自分でやると，思い込みの誤りが摘出できない．

• 原理115：ブラックボックスおよびホワイトボック

ス両方のテスティングを用いよ

いずれのテスティングも長短がある．いずれか一方だけでは徹底したテストはできない．

• 原理126：エラーを個人のものにするな

エラーを恥ずかしいものとして隠したのでは，それからチーム員が学ぶことができないから継続した改善はできない．

• 原理129：読んだことのすべてを信じるな

言い換えれば，事実の中に信じるべき真実がある，ということだ．これは自ら現実を調べた上で決定すべきことを言っている．

• 原理140：人員と時間は交換できない

これは，Brooksの不朽の名著，The Mythical Man-Monthにある有名な警句の1つである．

• 原理172：プロジェクトの事後検討会を実施せよ

これによって，チーム員はプロジェクトから教訓を学び，よりよいプロセスを目指す．DeMarcoは，プロジェクトの完了時でなく，各フェーズの完了時点で検討会を実施することを提案している．

• 原理178：すべての中間成果物に名称とバージョン番号を与えよ

これは，徹底したソフトウェア構成管理の要件でもある．

• 原理181：すべての変更を追跡し続けよ

これは，追跡可能な仕組みがあり，それにしたがってプロセスが行われていて，初めて可能になる．

• 原理194：最悪のコンポーネントを最初に作り直せ

最悪のコンポーネントは，後々までトラブルの原因となることが多いことが知られている．

• 原理196：変更した後には必ず回帰テストを行え

回帰テストは，重要であるという共通認識があるにもかかわらず，確実に実施されていない．その理由の多くは，それを継続的に支援する仕組みが欠けていることにある．テストケースやテストプログラムが個人の引き出しの中に非公式にしまわれていたのでは，回帰テストはその人がその部署を去ると共に消滅する．

• 原理199：最適化する前にプロファイラを使え

高頻度で通過するパスをプロファイラで調べてから，重点的に最適化を行えば，効果が大きい．

4.3 特定組織のベスト・プラクティス

実際に特定の組織で使われているベスト・プラクティスは，その組織のノウハウなので，目に触れることは少ないが，Microsoftで使われているものがいくつかの文献で公開されている（たとえば文献3），5），6））ので，それを紹介しよう．

(1)毎日システムを構築する

あるシステムの開発作業を分担しているすべての作業者は，機能性がまだ完全でなくても最後まで完全に走るものを毎日構築グループに提供する．提供物は管理者のすぐそばに集められるので，管理者からは，プロジェクトの状況が手にとるようにわかる．管理者は，これをプロジェクトの心臓の鼓動にたとえる．

(2)システムの要素に重大バグがあれば最優先で直す

もし，構築されたシステムがテストできなければ，バグの修正は最優先で行われる．しかし，テストに支障がない小さなバグは，寛大に扱われる．構築は，夜を徹して行われるから，テストに失敗すると多くの人に迷惑がかかるので，提供者の責任は重い．責任を果たすための提供前のテスト，あるいは構築後のバグ摘出のために，同じ部分を担当する相棒がおり，構築中に待機することがよくある．もし，待機していなければ，夜中の3時にバグでたたき起こされることもある．すべてのファイル，ライブラリ，および他のコンポーネントがうまくコンパイルでき，リンクでき，テストできないような重大なバグが存在しなければ，構築はうまくいったとみなされる．

(3)構築したシステム全体をテストする（煙探知器テストと呼ばれている）

システムに組み込まれた機能を端から端までテストする．最初のテストは機能が少ないので数秒で終わるほどの簡単なものだが，システムが進化するにしたがってテストは徹底したものになり，30分から1時間以上を費やすようになる．

(4)構築専任グループを設ける

たとえば，Windows NT 3.0の場合は，4名がフルタイムでこれを担当した．

(5)それをやるのが意味があるときだけシステムを修正する

ちょっと直せば全体テストが続けられるようなとき以外は，原則として，その場であわててプログラムを修正するようなことはしない．それは，数日ごとにまとめて，一貫性のある形で行う．

(6)構築を駄目にしたらペナルティを課す

プロジェクト要員は，健全な構築を続けることが最優先であり，構築の失敗はあってはならないことを最初に告げられる．もし，構築を台なしにした開発者は，ペナルティを課せられる．とはいえ，それはたあいのないもので，あるグループは，みんなにペロペロキャンディーを配り，他のグループでは山羊の角の付いた帽子をかぶり，あるいは5ドル払う．

(7)毎日の構築と全体テストは期限が迫っても実施する

期限が迫ると手抜きをしがちだが，このプラクティスはそれを許さない．

なお，これがIEEE Softwareに紹介された後，読者から，このテストに強く依存するプラクティスでは根本原因が改善されないではないか，という反論が寄せられた．

5. ベスト・プラクティスの導入

公式のモデルを用いたプロセス改善の導入は，通常あるべき姿，たとえば品質マニュアルの作成から始まるが，ベスト・プラクティスは社会科学や考古学でいうフィールド調査または発掘によって，何がベスト・プラクティスかを見い出すことから始まる．以下，順を追って述べる．

5.1 成功プロジェクトの識別

プロジェクト・メトリクスと直感によって，うまくいっているプロジェクトをいくつか選ぶ．

5.2 ベスト・プラクティスの発掘調査

何が成功要因であったかを，現実に行われたプロセスと作られたプロダクトを調べて探り出す．これは，作業者が意識しない場合もあるので，必ずしも容易でない．ときには，成功要因の仮説についてのグループ討議が有効なこともある．

5.3 ベスト・プラクティスの識別とまとめ

調査した時点でベストと思われるプラクティスを識別し，実施可能なルールにまとめる．識別したものがベストでなくベター・プロセスかもしれないが，それはあまり気にしない．

5.4 プロセスの変更と効果の測定

新たなプロセスの実施や変更に挑戦する．その結果によって，プロセスをさらに改善する，または，結果が悪くその理由がはっきりすればそれを放棄する，というダイナミックな適用を行う．そのためには，変化を恐れない挑戦意欲と，効果を正しく判定するための適切なメトリクスが重要である．ベストを選び固定的に実施するのでなくて，良さそうなプロセスをやってみて，結果を見ながらさらに良くしていく，というダイナミックな改善が，ベスト・プラクティスの真髄である．

5.5 プロセスの改善

変数が制御された計画的な実験により，プロセスと改善の関連を実証し，これに基づいてプロセスをより良いものにしていく．これなしには，ベストであることの根拠は主観でしかない．

5.6 ワースト・プラクティスを他山の石とする

一般に，ベスト・プラクティスを探るよりも，いくらでもある失敗プロジェクトからワースト・プラクティスを探る方が容易である．しかも，それを体系的にまとめれば，「べからず集」として活かすことができる．**5.1**～**5.3**は，失敗とワーストに置き換えて，そのまま実施できる．

6. ベスト・プラクティス実施上の落とし穴

(1)低いモラールと受け身の態度

繰り返し述べてきたように，ベスト・プラクティスはボトムアップ指向の改善活動である．したがって，開発の第一線にいる技術者の高いモラールに支えられた前向きの改善意欲，自発性，および，データに対する敏感さが不可欠の前提となる．組織にトップダウンの気風が充満し，組織の底辺が受け身に支配されていれば，成功はおぼつかない．

(2)トップの無関心

ボトムアップ活動とは矛盾するように見えるが，活動のトリガがボトムであっても，第一線技術者の活動をトップが支援しなければ活動は永続せず，失敗に終わるだろう．

(3)ベストの主観的な判断と安住

あるプラクティスがベストであることを主観的に判断をしがちである．もっと危険なのは，思い込んだベストや自己流に安住することである．

(4)不適切なメトリクス

ベターであることを検証するためには，適切なメトリクスがなければならない．たとえば，生産性において，再利用を適切に評価するメトリクスがなければ，ベストがワーストとして示されたり，またその逆も起こりうる．また，制御すべき変数がはっきりしないメトリクスも，プロセスの改善方向を見い出すことができない．なお，メトリクス制度の実施は，投資をともなうため，組織がトップダウンで実施すべきことである．

7. おわりに

実は，このアプローチを最初に耳にしたときの筆者の感想は，「これはかつて我々が通ってきた道ではないか？」であった．後になって，ほかにも同じ感想を持った人がいることを知ったが，まだプロセス改善という言葉が知られていなかった頃，日本のいくつかの組織は，品質サークルによるボトムアップの取り組みによって，ソフトウェアの品質問題を1つずつ地道に改善してきた．それを経験してきた人にとって，ベスト・プラクティスは，なじみのあるものと思うだろう．

ベスト・プラクティスそのものは，どこの組織にもどこかに存在する．ベスト・プラクティスは，それを発掘してその組織で普遍的なものにする活動である．これは，むしろ，日本の改善のアプローチに近く，そ

れだけに，実施しやすいとも言える．ただ，上で紹介したアメリカでの実例から見ると，追跡性，構成管理，および機能ポイント法のように，日本の開発現場では目だって適用が遅れているプラクティスが加えられている．それは，長い間，日本のソフトウェア産業が，ソフトウェア工学の導入を怠ってきたからであろう．

ベスト・プラクティスをさらによく理解したければ，IEEE Software誌の各号の最終ページに載っているコラム，ベスト・プラクティスを，また，近々Prentice-Hallから出版される予定の，Alan Davis監修のBook Series, Best Practices in Software Developmentを読むことを勧める．

参考文献

1) Paulk, M. et al.: The Capability Maturity Model: Guidelines for Improving the Software Process, 441p., Addison-Wesley (1995).
2) Bach, J.: The Immaturity of the CMM, American Programmer, Vol.7, No.9, pp.13-18(1994).
3) Yourdon, E.(松原友夫訳): プログラマーの復権(Rise and Resurrection of the American Programmer), 310p., pp.126-142,トッパン(1997).
4) Davis, A.(松原友夫訳): ソフトウェア開発201の鉄則（201 Principles of Software Development), 237p., 日経BP社(1996).
5) McConnell, S.: Daily Build and Smoke Test, IEEE Software, Vol.13, No.4, pp.143-144(1996).
6) Cusumano, M.A. and Selby, R.W.(山岡洋一訳)：マイクロソフトシークレット(Microsoft Secret), Simon & Shuster Inc., 日経新聞社(1995).

（平成9年8月27日受付）

松原　友夫(正会員)

1929年生．1950年早稲田大学専門部機械科卒業．1956年(株)日立製作所入所．1970年日立ソフト(株)設立と同時に入社．1991年定年退職．1992年から個人でコンサルタント業を営み現在に至る．1996年からソフトウェア専門誌IEEE Softwareの副編集長．現在同誌，American Programmer，およびInformation and Software Technology 3誌の編集委員を勤める．情報処理学会規格調査会SC7委員．Yourdonが主催するCutter Consortium指導員，ソフトウェア技術者協会，IEEE会員．e-mail:tmatsu@xa2.so-net.or.jp