機器組込み型ソフトウェアにおける開発管理の試み

キヤノン(株) 創造環境推進センター ソフトウェア技術開発部
馬場 健，中井 晶也，岩渕 洋一，小泉 毅[1)]

【要旨】ハードウェア・メーカーである当社とってはハード主体の開発工程・開発計画が優先しており，その中で年々大規模化する製品組込みソフトウェアの開発管理・プロセス改善は長年の懸案であった．多くの試行錯誤の末，'96 ころよりデータに基づいたプロジェクト管理手法の導入，ソフトウェア開発環境の整備という２つの観点から取り組みを行い，進捗・品質管理とソフトウェア開発者が設計に専念できる環境の提供といった面で成果をあげることができた．また２つの取り組みはソフトウェア・メトリクスの収集・蓄積を行うデータ計測環境という形で連携し，今後のより抜本的な改善への足がかりとなるものである．

1. はじめに

ソフトウェア開発における生産性・品質の向上は当社キヤノン(株)においても急務の課題であり，'90 ころより Products 5（生産性５倍）をスローガンに様々な取組みを行ってきた．

特にソフトウェア開発における開発管理・プロセス改善には我々としてもいくつかの施策を試みてきたが，ハードウェア主体の開発工程・開発計画を基本とした当社製品の開発現場にはなかなか立ち入ることができず，「本来こうあるべきだ」に基づくトップダウン・アプローチはことごとく失敗に終わったといえる．

このような状況の中で '96 ころより当社映像事務機開発センターと共同で「ソフトウェア生産性向上委員会」(通称 NSB: No Silver Bullet !) なるワーキング・グループを発足し，複写機系製品の組込みソフトウェアを対象に，

- データに基づくプロジェクト管理の導入
- ソフトウェア開発環境の整備

の２つの観点から新たな取組みを行った．

この２つは，当初独立に進行したものであるが，最終的にはまだ不十分ながらも「データ計測環境」という形で連携しつつある．

当時の映像事務機開発センターは３つの開発拠点への分散（大田区下丸子，茨城県取手市，静岡県裾野市）が決定され，特にソフトウェア開発においては同一製品・複数拠点での分散開発への対応と，また分散環境下での開発管理が急務の課題であった．

我々の目的はこれらの課題への対応とともに，これを契機として混乱に満ちたソフト開発全体を多少ともすっきりさせ，安定した開発を可能にすること，その上で今後のより抜本的な改善への方向性を見出すことである．

我々の個々の実施策自体は常識的で地道なものであるが，徹底的に実践的であることを目指した．ここでは開発管理・プロセス改善という難解な問題に向けて「まず何をなすべきか」という１つの事例として，我々の取組みを紹介したい．

2. プロジェクト管理の導入

プロジェクト管理に関しては

- 計画段階での定量的な見積もり
- 進捗チェックの仕組み作りと実施

を重点課題とした．

過去のデータ分析

まず過去の製品数機種のデータ分析を行った．

---

1) 現在経営情報システムセンター IS ソリューション企画部

機能量（機能的サイジングメジャー）についてはファンクション・ポイント法等いくつかの定義が知られているが，

- 組込みソフトにとっては算出の労力がかかりすぎる
- 開発者にとって直感的に理解できない

などの問題あり，「製品の機能仕様書・マニュアル等に記載されている機能項目の数」に「サービス・工場での調整用モードの数」を勘案して独自に算出することとした．

過去のデータは体系的に整理されたものでなく問題はあったが，機能量対総コード量，平均バグ密度などの基本的なパラメータについておおむね妥当な結果を得ることができた．（データ値は公表しないが，世の中と比べてかけはなれたものではない）

見積もり

これらのパラメータをベースライン[1]として，新機種の開発より以下の形を基本に定量的な見積もりを開始した．

- 機能量 → コード量
- 機能量 → バグ数
- 機能量 → 開発期間・工数
- コード量 → バグ数
- コード量 → 開発期間・工数

なお特に，開発期間・工数に関しては開発現場からは見積もり自体よりも進捗チェックの面からの要請が強く，以下の進捗管理を重点的に実施することとした．

進捗管理

進捗管理はソフト開発リーダが作成した作業項目の消化具合を示す「作業進捗管理表」と，定期的なコード量（NCSS）計測による「コーディング進捗」（図-1）を主体として運用している．

バグ数・期間・工数などの見積もり値はコード量の推移から適宜修正し，また評価が始まれば障害累積曲線から修正される．コーディング進捗等は後述のデータ計測環境により自動計測され，定期的に関係者に公開されている．

現在の状況

結果として程度の差はあるが，全開発機種について定量的な見積もりと進捗チェックが定常的に行われるようになった．特にハード最優先であった製品計画にソフトウェアの見積もりもかなり考慮されるようになりつつある．

また評価期間は製品計画上大きなウエイトを占めており，総バグ数の予測をもとに試作機台数や評価担当者の人員手配などの評価計画作成に特に貢献している．

また複雑度など他のメトリクスについても随時収集・分析を行っており，バグの偏在性（バグで修正したファイルや関数はまたバグのある可能性が高い）などについては明確な結果を得ているが，実際の運用という面ではまだ実験的段階にとどまっている．

3. 静的テストツールの導入

昨年の本シンポジウムでも報告されているが [2]，我々もほぼ同時期にスタティック・テストツールとして英 Programming

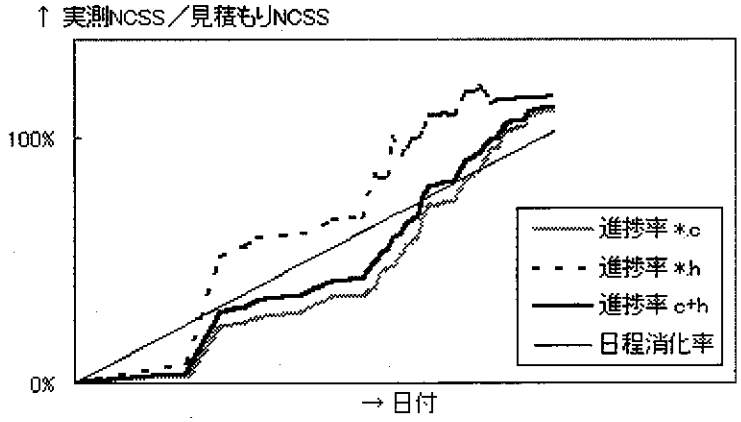

図-1　コーディング進捗

Research 社の QAC [3]の導入を行った.

特に組込みソフトウエアでは，スタティック・テストは実機ハードに依存なく行える数少ない手段の１つである.

警告の分類

支援内容は文献 [2]と大きく異なるものでなく，活用には QAC の指摘する多くの警告から「いかに有効な警告を効率よく抽出し，設計者に対策させるか」がポイントである.

我々は QAC の警告を「警告されたコードの危険度」と「そのコードの修正の難易度」から表-1 のようにレベル分けした.

表-1 QAC 警告のレベル分け

	危険度大	危険度中	危険度小
修正難	レベル０	レベル３	レベル６
中程度	レベル１	レベル４	レベル７
修正容易	レベル２	レベル５	レベル８

危険度大とはコーディングミスや論理ミスの可能性が極めて高いと思われるもの，危険度小とは例えば清書法に関するものなどである．移植性の問題や関数が大きすぎるなどはおおむね危険度中としている.

また修正容易とは特定行だけなど局所的なチェック・修正で済む場合，修正難はデータ型の修正など多数のファイルにまたがったチェック・修正が必要／あるいは修正すること自体に危険性があるなどである．例えば

- レベル０～ 境界調整数の厳しい型への型変換（これは非常に多い．I/O ポートを char で読んだり short で読んだりするためであろう）
- レベル１～ セットされない変数の使用
- レベル２～ 関数宣言と実引数の違い

などが代表的な警告である.

このような分類は実際に多数のソースでの警告の出現頻度と状況の分析を行い，「実際に使わせる」，そのためには「(理想論でなく) 実際の作業量が対応可能な量になる」ことに最重点をおいて作成した．このため「危険度大」は全体の警告数の 2～3%程度に押さえている．また「修正難易度」は「時間がなくても簡単なものなら直せるでしょ」，ということである.

このような分類は万能ではないが，実際のチェック・対策作業における目安として推奨ルールを作成し，またこれらをもとにコーディング・ガイドラインを作成した.

現在の状況

導入当初は解析依頼や問い合わせ等が頻繁であったが，現在は開発の進展具合と流用の程度に応じて各開発チームでの運用を基本としている．例えば

- あるチームではレベル２警告数 ＝ **0** をリリース基準として採用
- 開発の進んだ他のチームでは独自に特に危険な少数の警告を抽出して実施

などである.

また，このようなチェックと対策の実施は日常的に行うことが不可欠であり，上述の警告分類や推奨ルール等を次節で述べるソフトウェア開発環境の一部として組み込んだ形で提供している．また後述のデータ計測環境の一環として，開発チームでの運用とは無関係に定期的計測を実施している.

QAC 導入の効果についてはまだ定量的な結果は得られていないが，過去のバグ票では１～２割がスタティック・テストにより防止可能とされており，今後検証していきたい．ただまずは，「よいコードを書く知識・習慣を定着させる」という教育的効果に最も期待している.

4. ソフトウェア開発環境の整備

特に分散開発への対応とツール導入によるレベルアップを目指して，以下のようなソフトウェア開発環境の整備を行った.

機器組込みのソフト開発では，どうしても製品ハードウェアに合わせて開発チームが発足する傾向にあり，環境・ツール面でもチーム横断の統一した取り組みが遅れ勝ちである.

これはこれ単独で大きな課題であり，上記とは基本的には独立して実施した．なお開発マシンは HP-UX ワークステーションが主力である.

ソース管理

拠点間でのソースファイルの共有を実現するため，ファイルのバージョン管理ツールとして HP 社の Softbench/CM を導入した.

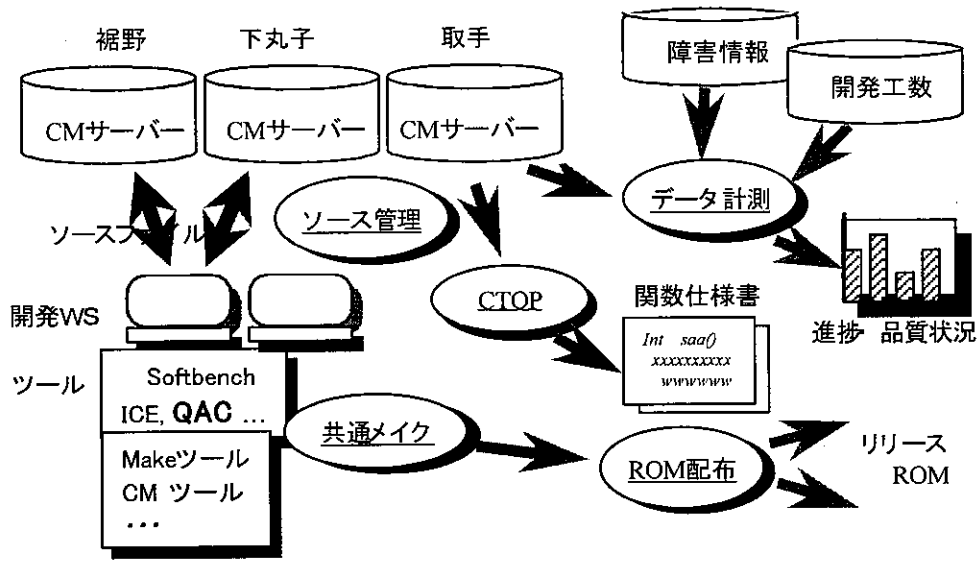

図-2 ソフトウェア開発環境とデータ計測環境

他のいくつかの市販ツールも検討したが，選定の理由としては分散開発に適した複数サーバーへのマッピング機能があることなどであるが，なによりもなれ親しんだ RCS のネットワーク対応版という感じで導入も使い方も簡単であることが大きい．

現在は３拠点のサーバーで全ソースファイルを管理しており，各開発者は必要に応じて他拠点のサーバーを自身の開発マシンにマッピングしてファイルを参照し，自身のファイルは自分の拠点のサーバーにチェックインする形で運用している．

また Softbench/CM は機能的には不足気味であり，構成管理・ブランチ開発等の多数の補完ツールを Perl で作成している．

共通メイク環境

ソース管理と同時に，機種間でまちまちであった開発ディレクトリ構成を統一し，機種間で共通使用できる各種のメイク・テンプレートを用意することにより，メイク（ビルド）の共通化を進めた．

確実で効率のよいビルドがいかに重要であるかは開発経験者には明らかであろう．

ソース管理と共通メイクは最大の課題であった分散開発に現在必須のものとなっている．また製品ソフトの再現性確保とともに，円滑なソフトウェア共有・流用を実現できた．

関連ツール

例えば上記の QAC 関連ツールもこの共通メイクに組み込まれており，開発者はワンタッチで必要なチェックを行うことができる．

またその他開発したツールとして，C ソースからのドキュメント・ジェネレータ CTOP(C To Pretty document：ちょうど javadoc の C 言語版）や評価用・製品用のリリース ROM データの管理・配布ツールなどがある．これらも図-2 のような形で随時共通メイクに組み込むことにより，作業の自動化を可能にしている．

またこれらの関連ツールや開発用のクロスコンパイラ／ライブラリ等もこの環境を通じて配布しており，ソース管理と共通メイクを中心とした基本的な開発環境サポートは，開発担当者が製品開発に専念できる環境を提供するとともに，新たな開発ツールの導入・普及のためのプラットフォームとしての機能を発揮している．

5. データ計測環境

まだ不十分な点は多いが，全体は上記ソフトウェア開発環境とそれを利用したデータ計測環境という形で図-2 のような環境を構成し，開発中の全機種について継続的なデータ収集と進捗・品質状況の提供を実施している．

表-2 主な計測データ

	主なメトリクス	元のデータ	計測・運用方法
ソース関連	NCSS QAC 警告数 ファイル修正回数	Softbench/CM	週1回深夜バッチで計測 Excel でレポート作成
障害関連	障害件数 評価工数	Lotus Notes	Notes SQL+Excel で随時オンラインで集計・レポート化
開発工数関連	開発工数	Oracle (全社システム)	不定期実施

主な計測データは表-2 のようであり，基本的にはそれぞれのデータベースを直接アクセスして計測・集計を行う手段を用意し，Excel でグラフ化・レポート作成している．

特にソースファイル関係は上記のソフトウェア開発環境と密接に関連し，ソース管理サーバーのファイルを直接チェックアウトして，共通メイクにより実際に使用されているファイルのみを正しく計測できる．

これらの計測作業は個々の開発チームとは離れて，生産性向上委員会の手で継続的に行われ，Web で関係者に公開されている．またチームからの要請に応じて進捗会議や品質会議の参考資料を作成・提供している．

全体の統合のレベルはまだ低く，特に計測したデータの管理や自動化は今後の課題である．しかし現在では，基本的な情報のほとんどが Perl CGI や Java servlet により Web で閲覧可能になっており，生産性向上委員会が作成した各機種の進捗・品質レポート，リリース ROM データのダウンロードやリリース履歴のほか，自動生成された関数仕様書やソース管理サーバー上のソースファイルそのものなども Web でアクセスできる．

6. 考察

現在ようやくいくつかの製品の出荷をむかえた段階である．生産性・品質に関する定量的な評価は今後の課題であるが，総じて従来より大きく向上したものではない．

はじめに述べたように我々の目的は早急な改善を目指さず，むしろその前段階としてソフトウエア開発環境や開発管理など最も基礎的な部分を整理し，今後の道筋を明確にしていきたいというものであった．

我々の取組みを振り返ってみると以下のような事項が言えよう．

何ができたか

上述の取組みの結果として，プロジェクト管理面では

- データに基づいた開発・評価計画立案
- 継続的なデータ収集とデータに基づいた進捗・品質チェック

が定着した．特に，ソフトウェアの進捗と遅延（多くの場合ハードトラブルの支援のためにソフト開発が遅延する）が多少とも見えるようになったことは大きい．

また開発環境・ツール面では

- ソースファイルやリリース ROM データといった成果物管理と製品再現性の確保
- データ収集やその他のツールの導入・自動化のためのプラットフォームの構築

が実現でき，分散開発への対応とともに，開発環境の保守等においても開発者の負担を大幅に低減することができた．

これらは連携して「見えないソフトを見えるように」という課題に対する情報提供手段として機能している．

今回の取組みは直接に開発プロセス改善を目指したものではなく，またトップダウン的に管理の導入を試みたものでもないが，以下のような事実から所期の目的のかなりを達成できたと考えている．

- 施策の多くは現場に受け入れられ，現在も継続的に実施されていること
- 製品開発チームと支援部隊の分離により，チームを横断した様々な問題・要望の収集と対応が可能になったこと
- これらの結果として，自然に成果物と情報が共通の資産として蓄積される仕組みがで

き，今後の取組みの上でのベースとなるものが構築できたこと

アプローチの方法

支援部隊としての我々の活動を振り返ってみれば，まずこの活動における最大の指針は「徹底的に実践的であること」であった．すなわち，

- 明らかに良いことは強く実施を推奨する
- 推奨したことは実施可能なレベルまでブレークダウンする
- 実施にあたっては良さを実感できるまであらゆる支援を行う

これらは例えばQACの導入においても，開発環境の整備や進捗管理においても，その導入・運用方法として色濃く反映されている．

また我々のアプローチの方法は（必ずしも意図したものではなかったが），結果的には以下のような徹底的なボトムアップ活動として特徴づけられるであろう．

- 開発現場の要望と現実の効果を最優先
- コーディング・評価の下流工程に集中
- 可能なかぎりツールでしばる

「(ルールでなく）ツールでしばる」などというのは，「少しでも楽したい」「ほっておくと元の木阿弥？」といった開発現場の風土を表している．

このようなアプローチの方法や個々の実施策のレベルには問題を残すものの，我々にとってはまず，当社のソフトウェア開発を管理可能なレベルに引き上げること，そのためには実際の開発現場に深く立ち入ることが必要であり，この課題に向かっての１つのベスト・プラクティス [5]であったと考える．

今回の取組みにおいては，手法とツール・環境両面からの支援が１つのキーとなり，また「開発拠点の分散」という現実の問題が実施にあたっての大きな契機となった．

7. まとめ

ソフトウェア開発管理の課題に対し，データに基づいたプロジェクト管理の導入と，ソフトウェア開発環境の整備という観点から取り組みを行い，所期の目的を達成することができた．

今回の取組みにおいては地道な施策を徹底して推進した結果，徐々にではあるが全開発チームに着実に浸透するとともに，データ計測環境など個々の施策以上の総合的効果を得ることができた．

今後については今回の成果をベースとして，現支援の継続・レベルアップとともに，欠落が明白である上流工程・その他より本質的な問題に取り組んでいきたいと考えている．

謝辞

本取組みにあたってご協力いただいた日本ヒューレット・パッカード株式会社，株式会社東陽テクニカの方々に感謝いたします．

また貴重な機会を提供いただき，ご協力いただいた当社映像事務開発センターと関係部門の方々，およびソフトウェア技術開発部の方々に感謝いたします．

参考文献

[1] 大場 充：「ソフトウェア・プロジェクトの実績データ収集・分析技法」，ソフト・リサーチ・センター，1993

[2] 小笠原他：「プログラム静的解析の適用事例と効果分析」，ソフトウェア技術者協会 ソフトウェア・シンポジウム'98 論文集

[3] QAC については以下など
http://www.toyo.co.jp/ss/products/qac.htm

[4] Softebench/CM については以下など
http://www.hp.com/esy/go/softbench.html

[5] 松原 友夫：「ベスト・プラクティスの思想と実践」，情報処理，V0l.39, No.1, pp.43-48 (1998)