レガシーシステムからのシステムマイグレーション

― JFE スチール知多製造所生産管理システム再構築を通じて

JFE システムズ株式会社

■ 執筆者Ｐｒｏｆｉｌｅ ■

檜山 直

1982 年　川崎製鉄株式会社入社
　　水島製鉄所 システム部システム室配属
2005 年　現在 JFE システムズ株式会社
　　中部事業所所属

小島 正人

1981 年　川崎製鉄株式会社入社
　　水島製鉄所 システム部システム室配属
1998 年　川鉄情報システム株式会社
　　中部事業所配属 鉄鋼グループ担当
2005 年　現在 JFE システムズ株式会社
　　中部事業所所属 開発グループ担当

田中 公資

1982 年　川崎製鉄株式会社入社
　　知多製造所 企画部システム室配属
2005 年　現在 JFE システムズ株式会社
　　中部事業所所属 技術グループ担当

東 睦仁

1992 年　川鉄情報システム株式会社入社
　　知多事業所配属
2005 年　現在 JFE システムズ株式会社
　　中部事業所所属 開発グループ担当

■ 論文要旨 ■

JFE スチール知多製造所では，従来，メインフレームで稼動してきた鋼管生産管理システムの硬直化と高コストを解決するために，Unix サーバをプラットフォームとするオープンシステムへのレガシーマイグレーションを行った．レガシーマイグレーションは単純にハードウェアや OS といったシステム基盤の置き換えだけではすまない．基盤の変化はその上位にある実装技術・アプリケーション設計・開発方法論にも影響を及ぼす．JFE スチール知多製造所のシステムリフレッシュでは，要求仕様を出す実務ユーザーとシステム開発者にとってそうした影響が大きな障害にならないように，現実的な技術や手法・方法論を選択した．その結果，完全オープン化による大幅なシステム費用削減と，納期短縮や在庫削減をはじめとする実務効果を実現し，さらなる業務改善活動の基盤を提供した．

■ 論文目次 ■

■ 図表一覧 ■

1. はじめに

JFE スチール知多製造所は愛知県半田市に位置し，主にシームレス管・溶接管などの鋼管製品を製造している．そこではコンピュータシステムとして，これら製品の生産管理システムが稼動しているが，1998 年から約 5 年をかけて生産管理システムの，データベース構造の変更やアプリケーションを含む全面的なリフレッシュを行った．これは従来メインフレーム上で稼動していたシステムを Unix サーバ上に再構築するもので，いわゆるレガシーマイグレーションとしての側面を持つ．

一般にレガシーマイグレーションはハードウェアのリース費用・保守費用，ソフトウェアのライセンス費用の削減を狙って行われる．実際にメインフレームでのレガシーシステムと Unix や Windows などのオープンシステムを純粋に比較した場合，これら費用の差は一目瞭然である．実現すれば確実な効果が得られることはわかっている．

反面，レガシーシステムの資産の移行にかかるコストが見えづらく，それが得られる効果に見合う範囲に収まるのか，そもそもレガシーマイグレーションは実現可能なのか，といった疑問がそこに立ちはだかる．JFE スチール知多製造所で行ったようなシステムの再構築ではなく，単純なコンバートにとどめるにせよ，ハードウェアや OS・DBMS・運用管理ツールなどのインフラが変わる以上，アプリケーションへの影響は避けられない．それは単に技術的な変化にとどまらず，アプリケーションを作成する人の考え方(例えば設計手法や開発方法論)までを変えていく必要にせまられることもある．こうした変化に追随するコストの見積もり方法が確立しておらず，その効果が見えているにもかかわらず，レガシーマイグレーションに踏み切れないケースも少なからずあるだろう．

本稿ではレガシーマイグレーションを検討する際の一つの目安として，JFE スチール知多製造所生産管理システムの再構築において，求められる変化にどのように対応し，結果的にどういった効果を得ることができたかを論じていく．

2. 知多製造所生産管理システムリフレッシュ

2.1 旧システムの課題

知多製造所では 1966 年に最初のコンピュータシステムが稼動し，1977 年までにオーダエントリ，命令システムなどのバッチシステムがまず構築された．そして 1983 年の小径シームレス管工場にはじまり，工場単位にオンラインの操業システムが順次立ち上がっていき，1992 年には基幹システム全体が確立した(表 1)．

このように各システムを順々に構築してきたという経緯の中で，次のような弊害が見られるようになった．

(1) システム資産の増大

それぞれのシステムで同様の機能を個別に作成したため，必要以上にシステム資産が増大し，それぞれのシステムに類似するアプリケーションやデータベースが散在することとなった．これは単純にアプリケーションやデータベースの数を増やすと同時に，システム間連携を複雑にする弊害を生じさせた．

表 1: JFE スチール知多製造所生産管理システム開発の経緯

暦年	開発システム
1966	パイプ生産月報
1974	パイプ 1 次
1977	パイプ 2 次
1980	設備管理
1983	小径シームレス，シームレス計画 I 期
1984	シームレス素材，小径シームレス管理 I 期，中径シームレス(操業)
1986	中径シームレス
1989	鋼管物流，技術標準管理
1990	溶接管操業(1 次，2 次，3 次)，小径シームレス圧延順序組
1991	溶接管操業，特殊管
1992	フレキシブル管

(2) システムの硬直化

複雑化かつ増大したシステム資産は機能追加や改善を困難にする．更に世代交代などによりシステムのブラックボックス化が進んでいった．そのため実務部門からの業務改善要求に対し，情報システムの硬直性がネックとなって，迅速に対応できなくなっていた．

(3) システム費用の高止まり

上述の状況を改善しようとしても Cobol での開発生産性がネックとなり，劇的な変化が望めない．更にメインフレームのハード費用やメンテナンス人件費も容易に下げられず，システム費用が JFE スチールの他地区工場と比較して割高になっていた．

これらの弊害を解消すべく，1998 年から以下のようなスケジュールで Unix サーバをベースとするオープンシステムへの全面リフレッシュを行った．

I 期　(1998-1999)　現品情報の一元管理とリアル化．
II 期　(1999-2000)　管理系システムのダウンサイジング．
III 期 (2001-2003)　操業系システムのダウンサイジング．
IV 期　(2004-)　　リフレッシュ基盤上での効果発揮期．

2.2 システムリフレッシュの狙い

システムリフレッシュにあたっては，以下の実務上の狙いがあった．

- 知多生産管理の全スコープ(オーダ投入から出荷まで)にわたるデータ分析を行い，上記スコープにおいて現品(一品)とオーダとの紐付き関係を一元的に管理するデータベースを構築する．
- 上記で構築された一元化データベースを活用することにより，従前のロット生産管理の弊害を廃し，一品紐付き生産管理を可能とし，納期短縮・在庫削減・歩留まり向上・物流費削減などの実務効果発揮の基盤を構築する．

また，システム面からは，メインフレームから Unix サーバをベースとするオープンシス

テムに移行することにより，大幅なハードウェアコストの削減とオープンシステム環境に基づくシステム開発生産性向上，及びメンテナンス生産性向上による開発・メンテナンス人件費の削減を狙いとした.

2.3 新システムの特徴

2.3.1 システム構成

新システムは表 2 に示すハードウェア・ソフトウェアによって構築された．オープンなマルチベンダの構成とすることで適材適所の選択が可能となり，全体としてのシステム費用を抑制することができた．更に将来にわたって構成の変更・拡張も旧システムと比較して容易になった.

表 2: 新システム構成

CPU	Unix サーバ，PC クライアント
OS	Solaris，Windows
DBMS	Oracle
開発言語	PL/SQL，Visual C++
運用管理ツール	SystemWalker

2.3.2 データベース一元化

旧システムの弊害の主たる要因となっていたアプリケーションやデータベースの重複を排除するため，新システムでは最初に従来例の見ない広範囲でのデータ分析を行い，データベースの一元化を行った．その中でも全品種の素材から製品に至るまでの工程を一元的かつリアルタイムに管理するテーブル群をマスターDB と呼ぶ．新システムのアプリケーションは，各々のアプリケーションごとに部分最適化された目的別データベースを構築するのではなく，上記の一元化データベース(マスターDB)を参照・更新する形で構築した．これによりアプリケーションの構造がシンプルとなり，システム間の連携の複雑さも解消することとなった.

3. レガシーマイグレーションの実際

3.1 オープンシステムにおける変化

オープンシステムに移行すればレガシーシステムでの弊害が解消することはわかっていた[1]．しかしながら移行を実現するには，ただハードウェア・ソフトウェアを置き換えるだけではすまない．アプリケーションの開発方法論・設計手法・実装技術も基盤の変化に合わせて変えていく必要に迫られる．これらは人が絡んでくる部分だけに，単純にツールのたぐいを導入すればすむ問題ではない.

例えば知多製造所の生産管理システムについて言えば，アプリケーション開発に直接関わるシステム基盤で以下のような変化が起こった.

[1] 大和田[2004]，pp.50-75.

DBMS　　：ネットワーク型 DB → リレーショナル型 DB

開発言語：Cobol(手続き型) → SQL(非手続き型), Visual C++(オブジェクト指向)

これらの新しい要素技術は，しばしばアプリケーションの作り方と結びついて高生産性の実現が謳われる．開発言語を Cobol から C++にしたからといって，言語の置き換えだけでは生産性向上を説明できない．デザインパターン[2]などの実装技術，モデリングなどの設計技術，更には開発方法論までを伴って，はじめて開発・保守の生産性向上のストーリーを描けるわけである．

3.2 変化への障害

これら新しい技術や考え方はすでに確立されたものであるから，新しいシステムに適用することは当然可能である．しかしながらその変化に人がついていけない．Cobol 技術者が C++の文法を理解することは難しくないが，オブジェクト指向の考え方を身に付けるのは容易なことではない．

もちろん最近でははじめから新しい技術・考え方を身に付けた若い技術者も入ってきている．だが彼らは逆に生産管理の業務知識が乏しい．新しいシステムを構築するには，旧システムで培われたメインフレームレベルの運用管理のノウハウや業務知識を持っている技術者が欠かせない．前述のとおりオープン系の技術は実装・設計・開発方法論が関連しており，メインフレーム技術者と若手で役割分担をすればよいというほど単純な話にもならない．

このような状態が根本的に変えられないとした場合，どうすればレガシーマイグレーションを進めることができ，またそれでどの程度の効果が得られるものなのだろうか．

3.3 知多製造所生産管理システムの場合

3.3.1 実装

(1) バッチ処理

新しいシステムではバッチ系のアプリケーションを PL/SQL で開発した．もともとバッチ系の処理は Cobol のような手続き型の言語との相性が良い．PL/SQL も「Procedural Language Extensions to SQL」という名前が表すように基本的には手続き型の言語である．したがってアプリケーションの作りは Cobol と大きく変わらず，メインフレームの技術者にとっても習得しやすいものである．ただしデータベースへのアクセスについては組み込みの SQL 文を利用するため，ここは従来の Cobol とはかなり違ってくる．

データベースアクセスにおける SQL の生産性の高さには定評がある．さまざまな集合演算・関係演算がサポートされていて，一つの SQL 文でネットワーク型データベースでは考えられないような複雑な処理を実現できる．しかしながら SQL が持つ機能をすべて習得するのは容易ではない．更に SQL では，書き方によってパフォーマンスが全く違ってくるため，そうした面まで考慮して最適な SQL 文を書くことは，メインフレーム技術者にとって大きな障害となる．

そこで SQL については次のような進め方で開発を行った．

---

[2] ガンマ[1999].

- SQL は比較的単純なデータ操作文(DML)を利用する.
- 典型的な DB アクセスについては SQL のサンプルを用意する.
- 上記サンプルを組み込んだ Cobol ライクな PL/SQL の雛型を提供する.

これではせっかくの SQL の生産性が生きてこないのではとの懸念もあったが，データベースアクセスのためのサブルーチンを個別に作っていた旧システムの開発と比較すると，単純な SELECT 文でさえ遥かに柔軟に，しかも簡単にデータを検索することができる．これは更新処理についても同様である．Cobol プログラムと行ごとに対称になるほど Cobol ライクな雛型を用意したため，メインフレーム技術者に対する教育のコストも思いのほか小さく抑えることができ，加えて SQL を利用するメリットも享受できた．それと引き換えにコードは冗長になったが，可読性を落とすものではない.

(2) 会話処理

クライアントで動作する会話アプリケーションは Visual C++で開発した．Visual C++は C++で Windows アプリケーションを開発するための統合開発環境(IDE)である．一般に C++はカプセル化・継承・多相性などの特徴によって，高い再利用性と保守性を実現すると言われている．しかしながらそうした特徴を利用するには，実装以前にしっかりとしたモデリングに基づくクラス設計を行う必要があるが，これは Cobol での開発とは全く異なる設計方法であり，メインフレーム技術者にとってはなじみにくいものである．当初は全面的なオブジェクト指向開発も検討したが，相性が良いと思われる操業系のトランザクション処理をオブジェクト指向ベースで開発し，そのほかの機能については次のような考え方にもとづいて開発を進めた.

- Cobol で提供されていたレベルの共通部品(e.g. DB アクセスなど)を用意する.
- 画面作成については IDE の機能を利用する.
- 業務ロジックについては Cobol と同様に手続き的に記述する.
- 上記サンプルを組み込んだ雛型を提供する.

C++はもともと C との互換性を考慮しているため，オブジェクト指向的な機能を使わなかったからといって，記述が不自然になることはない．それよりもメモリ周りの低レベルな管理によるメモリリークなどの問題の方が重要である．データベースへの問い合わせ結果など大きな動的メモリを必要とする処理については，メモリ管理を自身で行う共通部品を提供し，アプリケーション開発者がメモリの確保・解放を意識しなくてすむようにした.

この進め方で開発効率がどう変わるかだが，

- データベースアクセスには SQL を利用するため，SQL の高い生産性が期待できる.
- 業務ロジックの実装効率は従来の Cobol レベル.
- 画面作成は IDE のビジュアルな編集機能により大幅に向上.

のようにオブジェクト指向にこだわらなくてもかなりの改善が達成でき，オブジェクト指向開発を適用した部分も期待どおりの効果を上げた．とくに画面が簡単に作成できるようになって，後述するプロトタイピング開発にうまく適合したことが，開発全体の生産性を押し上げた．

3.3.2 データベース/アプリケーション設計

「新システムの特徴」で述べたように，新しいシステムではまずデータベースを一元化すること，言い換えればデータ構造を規定することから開発がはじまった．かつてフレデリック=ブルックスは『人月の神話』の中で，

> 私にフローチャートだけを見せて，テーブルは見せないとしたら，私はずっと煙に巻かれたままになるだろう．逆にテーブルが見せてもらえるなら，フローチャートはたいてい必要なくなる．それだけで，みんな明白にわかってしまうからだ．[3]

と書いているが，我々も一元的なデータベースを構築すれば，自ずとそれを利用するアプリケーションの形が決まると考えた．

データ構造をまず規定する設計手法といえば，たとえば T 字形 ER[4]のようなデータ中心設計(DOA)が想起される．しかし新しいシステムでは DOA の考え方を必ずしも厳密に取り入れてはいない．データベース設計における正規化は厳密ではなく，テーブルレベルの重複は排除したものの，項目レベルの重複は依然として残っている．

これには大きく二つの要因がある．

(1) 複数のプロセス複数のサブシステムに跨る広大なデータ分析の困難性
システムリフレッシュの対象スコープが広範で，従来のボトムアップ的な DOA を適用するには範囲が広すぎた．そこで対象の全スコープに関して知識のあるごく一部の要員が，トップダウン的なデータ分析と現行システム有識者との擦りあわせを行ったが，現行システムのブラックボックス化した状況もあいまって困難を極めた．これに関しては我々もアプローチ方法の正解を得ていない．

(2) 現行システムとの妥協
当然のことながら新システムはグリーンフィールドに構築されたものではない．既存のシステムが存在しており，新システムの立上げにおいてはリスク分散の観点から段階的な立上げを選択せざるを得なかった．したがって新システムを現行システムから切り離して考えることは不可能である．またシステムリフレッシュに際して，ユーザーが新システム構築のための広範囲なシステム要件を定義することは困難であり，今回の場合も「現行のシステム機能を保証してくれればよい」ということになる局面が多かった．ここに「現行システム保証」という要件が発生する．この「現行システム保証」という要件が，今回のシステムのテストフェーズ及び立上げフォローフェーズを通じて最大のボトルネックとなった．同様なシステムマイグレーションを考える場

[3] ブルックス[1996]，p.95.
[4] 佐藤正美[1999].

合にも大きな課題といえるだろう.

それでも品種・プロセスを超えて，データを一元的に，かつリアルタイムに管理できるようになったメリットは大きく，リアルタイムに更新される一元化ＤＢの実現が，実務効果発揮につながった.

3.3.3 開発方法論

メインフレームで稼動していた旧システムはウォーターフォール型の開発方法論にもとづいて開発された．ウォーターフォール型の開発では下流工程になるまで，ユーザーにシステムの具体的な形が見えないという問題がある．今回の開発ではユーザーが広範囲な既存システム仕様を，新 DB を前提に，一貫整合的に整理する困難さ，及び，基盤となる技術が変わることもあって，ウォーターフォール型の開発では下流工程で大きな手戻りが発生することが危惧された．ウォーターフォール型とは異なる開発方法論の必要は早くから認識していた.

そこで適用したのはプロトタイピング型の方法論である．これは開発の早い段階からプロトタイプを作成し，ユーザーとの合意を取りながら開発を進めていくものである．実装や設計手法での

- SQL や IDE といった技術を利用することでプロトタイプの作成が容易になった.
- 最初にデータベースを一元化しデータ構造が規定されたことで作成すべきアプリケーションの形が見えやすくなった.

という変化とあいまって，プロトタイピング型の開発は機能し，危惧された下流工程での手戻りを抑止できた.

4. 知多製造所生産管理システムリフレッシュの効果

これまでも個別には言及しているが，最後に新システム構築によって実現できた効果をまとめる.

まずシステム面から見ると，オープンシステムへの全面的な移行によりハードウェア費用を大幅に削減させることができた．更に一元化データベースの構築によりデータベース・アプリケーションの重複が解消され，システム資産が劇的に圧縮された．もちろんシステム資産の圧縮はメンテナンス要員の抑制につながる．加えて新しい技術(開発方法論・設計手法・実装技術など)による開発・保守生産性を向上させたことで，ハードウェア・ソフトウェアの両面において，システム費用の高止まりという課題を解決した．また一元化データベースの構築はそれを利用するアプリケーションをシンプルにし，その透明性を高めた．結果として実務部門からの改善要求に迅速に対応できるようになった.

一方，実務面での効果を見てみると，従来のロット単位の管理から一品管理を可能にしたことにより，納期の短縮，在庫の削減，歩留まりの向上，物流費の削減などの改善につながった．またリアルタイム性の高い一元化データベースが構築されたことにより，生産データの解析が容易となりスタッフ部門における生産性向上を実現した.

このように旧システムが抱えていた課題は，実務・システム両面においてほとんどが解消された．更に新しいシステム基盤を前提とした，更なる業務改善活動が開始され，着実な成果を発揮しつつある．

5. おわりに

繰り返しになるが，レガシーマイグレーションは単純にハードウェアやOSといったシステム基盤の置き換えだけではすまない．基盤の変化はその上位にある実装技術・アプリケーション設計・開発方法論にも影響を及ぼす．しかしながらシステムを開発する人がその変化に速やかに対応できるとは限らない．それはシステム技術者ばかりでなく，要求仕様を出す実務部門の担当者についても言えることだ．

「レガシーマイグレーションの実際」で見たように，知多製造所生産管理システムリフレッシュではメインフレーム技術者が容易にオープン系システム開発に入れるように，効率の良い最新の技術を必ずしも適用していないが，旧システムと比較すれば確実な向上を実現している．そして何よりも資産圧縮の効果が大きい．結果的に開発規模×開発生産性で表される開発費用は劇的に削減された．

もちろん何の工夫もなしにオープン化を実現できるものではないが，決して新しい技術へのスキルコンバートの難しさを理由に，レガシーマイグレーションの動きを止めてしまうべきではない．果たすべきはレガシーからオープンへの移行であり，最新技術の適用ではないのだから．

知多製造所の情報システムは，今まで述べてきた経緯にてリフレッシュされたが，一旦は活性化された情報システムも，改善活動がストップしてしまえば，長年後には旧システムのようにいつしかシステムのブラックボックス化が進み，再び「現行保証」がシステム要件となってしまう恐れもある．レガシーマイグレーションに伴う大規模開発あるいはシステムリフレッシュが完了してからも，業務改善～システム改善のサイクルを止めることなく継続的に回転させ，情報システムとともにシステムをメンテナンスするシステム要員が活性化された状態を保っていかねばならない．

参考文献

大和田尚孝[2004]，「さらばレガシー」，『日経コンピュータ』2004年6月28日号，東京，日経BP社．

ガンマ，E. et al.[1999]，『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』，東京，ソフトバンクパブリッシング．

ブルックス，F. P.[1996]，『人月の神話―狼人間を撃つ銀の弾はない』，東京，アジソンウェスレイパブリッシャーズジャパン．

佐藤正美[1999]，『T字形ER データベース設計技法』，東京，ソフトリサーチセンター．