レガシーモダナイゼーション　リホスト　メインフレーム

情報システムの継続的利用のためのレガシーモダナイゼーション

大規模なメインフレームシステムは複雑化やブラックボックス化が進み，保守開発やシステム更改の費用が高い状況が続いています．オープンシステムへの移行によって費用を抑えようとしても，全く異なるプラットフォームへの移行は容易ではありません．本稿では，大規模メインフレームシステムのリホストによるオープン化の難しさと，システム開発の自動化を活用した打開策について説明します．

谷野（たにの）　英昭（ひであき）

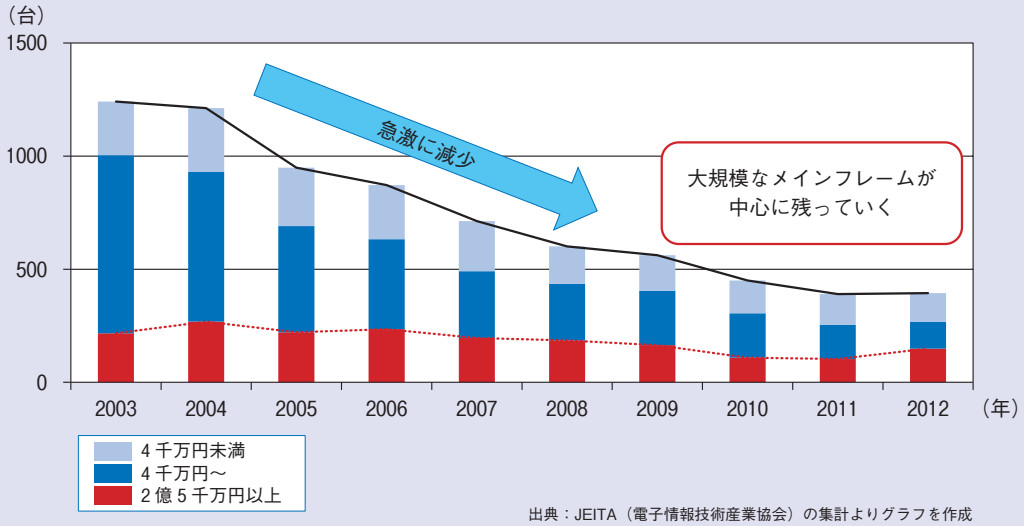
NTTデータ

残された大規模メインフレーム

1980年代から全盛期を迎えたメインフレームは，ピーク時には国内の出荷台数で約3500台，出荷金額は１兆円を超える規模となりました．しかし，1990年代中ごろからはオープン化の波にのまれ，その規模は10分の１以下にまで縮小しました（**図１**）．特に，中小規模（１台2.5億円以下）のメインフレームは急激に減少しました．このままメインフレームはオープン系のコンピュータに置き換わるかに思われましたが，ここ１，２年で減少ペースが止まってきました．大規模（１台2.5億円以上）なメインフレームは増加の傾向もみえます．これは，当面の間はメインフレームが継続して利用されることを示しています．

レガシーシステムの問題

NTTデータは，残されたメインフレームシステムを国内でもっとも多く運用している企業の１つであるため，メインフレームに関する多くの問題に直面しています．もっとも顕著なのは，システムの運用にかかる費用が高いという問題です．その原因は，ハードウェアが高価なだけでなく，長年利用して改造を加え続けたアプリケーションが肥大化・複雑化し，小さな変更が発生した場合でも膨大な影響範囲の調査とテストの実施が必要となるためです．特に，日本のメインフレームで運用されているシステムは大量の帳票を出力するバッチ処理が大規模・複雑であり，きめ細やかな業務処理を実現するためにアプリケーションも細部までつ



出典：JEITA（電子情報技術産業協会）の集計よりグラフを作成

図１　メインフレームの規模別出荷台数の推移

くり込みをしてきた経緯があります．さらに，2007年問題によって，大量のメインフレーム技術者が引退したため，システムのブラックボックス化が進み，問題を深刻にしているのです．

リホストによるオープン化

メインフレームを使ったシステムの運用の費用を抑えたいというニーズにこたえるため，一躍脚光を浴びてきたのがリホストと呼ばれる開発手法によるオープン化です．リホストでは，メインフレームで使われていたアプリケーションや処理の順番を記述したJCL（Job Control Language）やジョブネットをオープンシステム上で動作するように変換して再利用します．今まで使ってきた資産をそのまま利用して，早く，安く，品質を変えずにオープンシステムに移行することをねらっています．当初は，変換するアプリケーション数が数百程度でしたが，対象システムも大規模化し，アプリケーション数が1万を超える大規模システムにもリホストによるオープン化が取り組まれるようになりました．

このころから，リホストの費用や期間が当初想定を大幅に超えて問題化する例が多く聞かれるようになりました．調査をすると，もっとも大きな原因はアプリケーション変換に対する過信でした．変換さえしてしまえば新しいシステムがつくれるという誤解により，再設計によるインテグレーションが必要な点をおろそかにしていたのです（**図2**）．再設計が必要となるのは，主にバッチ処理やシステム運用，アプリケーション基盤の部分ですが，その規模はバッチ処理だけでJCLやジョブネットの数は数万本にもなります．その膨大な数の1つひとつに対して，処理順序や起動条件，処理やバックアップの時間，失敗したときのリカバリ方法などを考慮しながらシステム全体を正しく動作させることは針の穴に糸を通すような繊細な作業なのです．これらを非機能要件が異なる新しいプラットフォームで動かそうとすると，プログラム単体のテストでは問題なかったものが，結合テストをしてみると大量に不具合が起きるのです．

もう1つの大きな原因は，現状把握の難しさによる見積りの精度の低さです．過去20～30年をかけてつくり込んだシステムは，その場しのぎでイレギュラーにシステムを変更しているものが大量に組み込まれています．システムの一部の領域だけを調査しただけでは，イレギュラーケースやリスクの見落としによって，精度の高い見積りができないのです．最悪の場合はすべてのアプリケーションを1つひとつ確認する必要があるのですが，そのための費用と期間によってリホストの魅力を失ってしまうのです．

N字型による開発の自動化

打開策はアプリケーションを変換して再利用することではなく，信頼できるソースコードなどの資産を使って設計書を作成し，その設計書を再利用することです．はじめに，正しい設計書を手に入れることで現状のシステムの仕様を正確に把握し，次に最適なシステムを実現するためにシステム開発を徹底的に自動化して高速に開発するのです．この開発プロセスは通常のV字型の開発プロセスの前段に，再利用する設計書をつくるプロセスを加えたN字型となるのです（**図3**）．

N字型開発の最初は復元フェーズであり，現状の業務とシステムを可視化して理解するところです．そこには，リバースエンジニアリング技術をフル活用しています．ビジネスロジックとアプリケーション構造の可視化はコンピュータで自動的に実施できるためです．次に，人間が業務とシステムを関連付けて理解するために，可視化した構造やデータに業務的な意味付けをします．オンライン処理であればシステ

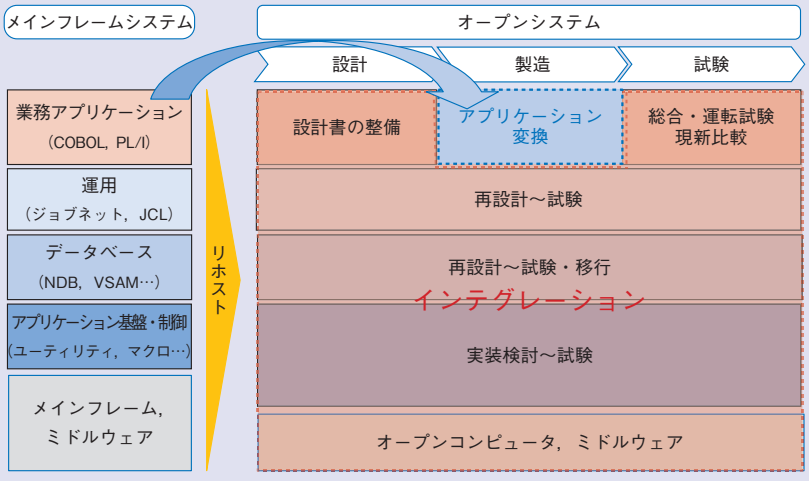


図2　リホストで見落としやすい「インテグレーション」

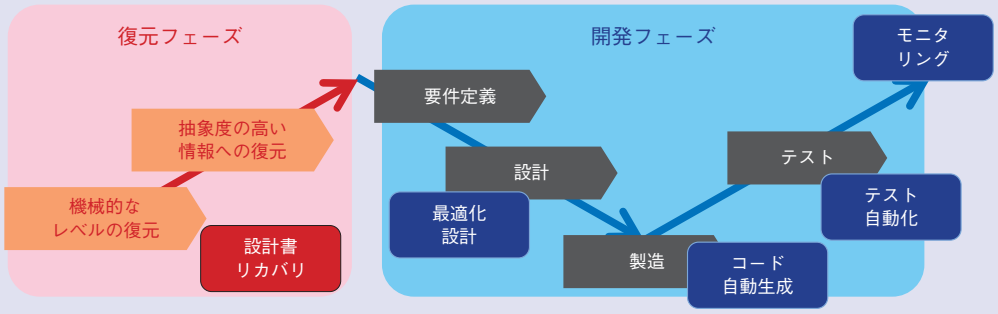


図３　Ｎ字型開発

ムの利用者が扱う画面から，実際に処理を動作するアプリケーションやデータベースまで一貫したトレーサビリティと意味付けをします．バッチ処理では，人が認識をしやすい処理の塊の単位に意味付けをします．処理の塊の単位は複数階層にして整理し，上位階層では業務用語を使って名称や説明を付与します．この業務とシステムの理解のプロセスによって，システム全体の俯瞰と詳細な把握が可能になります．

処理の理解と同時に，データの理解も必要です．システムで扱うデータの１つひとつに正しい名称と内容を定義するためには，データどうしの関係を整理して静的な構造を把握した後に，データの導出ロジックを明確にする必要があります．その際にも，リバースエンジニアリングの技術が利用できます．CRUD図*を自動でつくることで，データの導出や変更ロジックを特定することができるのです．このフェーズの最終ゴールは，システムのつくりに依存しない，純粋な業務の仕様を明らかにすることです．業務仕様を明らかにできれば，次のシステムの設計において現状のシステム構造や処理方式にとらわれずにあるべきシステムの姿が設計できるのです．

Ｎ字型開発の後半は，開発フェーズです．ここでは，前段の復元フェーズで作成した成果物を次期システムの上流設計のインプットとします．インプットの形式を標準化することで，システム開発の自動化にもスムーズにつなげることができます．NTTデータで開発したTERASOLUNA ViSCやIDEによるソースコードの自動生成やTERASOLUNA RACTESによるテストの自動化だけでなく，さまざまなツール群を活用した高速開発が実現できます[1]．新たな要求や最適化を行う際には，TERASOLUNA DSを使って設計フェーズから整合性を自動的に確認します．TERASOLUNA Simulator[2]を使えば仕様の妥当性までも早い段階で精度高く確認ができるのです．設計書を再利用するN字型開発によって自動化ツールを最大限活用し，システム更改をもっとも早く，確実な開発を実現できます．

最適化と再レガシー化防止

リホストとは異なり，Ｎ字型開発はシステムを再設計するため，現状のシステムが抱えている問題を解決することができます．大規模メインフレームシステムが抱えている問題の１つとして，大量のバッチ処理によってオンラインのサービス時間が限定され，リアルタイムで業務状況を把握しにくい点があります．これは，開発当時ではコンピュータパワーが不足していたため，応答時間を重視するオンラインで処理をさせず，できるだけ夜間のバッチ処理で対応してきたためです．できるだけオンラインでリアルタイム処理するためには，情報をつくり出すための業務仕様から整理をして処理を設計する必要があり，Ｎ字型開発による再設計による最適化が必要となるのです．データ管理についても大きな問題があります．メインフレームシステムでNDB（Network DataBase）が使われている場合は，オープンシステムではRDB（Relational DataBase）に変更する必要があります．その際に，NDBの構造を色濃く残したままのテーブル構造にしてしまうと，テーブルの結合やデータアクセスが頻繁に発生してしまい，効率の悪いデータベースアクセスによる性能問題を引き起こしてしまいます．Ｎ字型開発であれば，業務要件から最適なテーブル設計を行うため，データ構造が引き起こす問題

＊ CRUD図：データが，どのアプリケーションで作成（Create），参照（Read），更新（Update），削除（Delete）されるかを表形式で表現したもの．

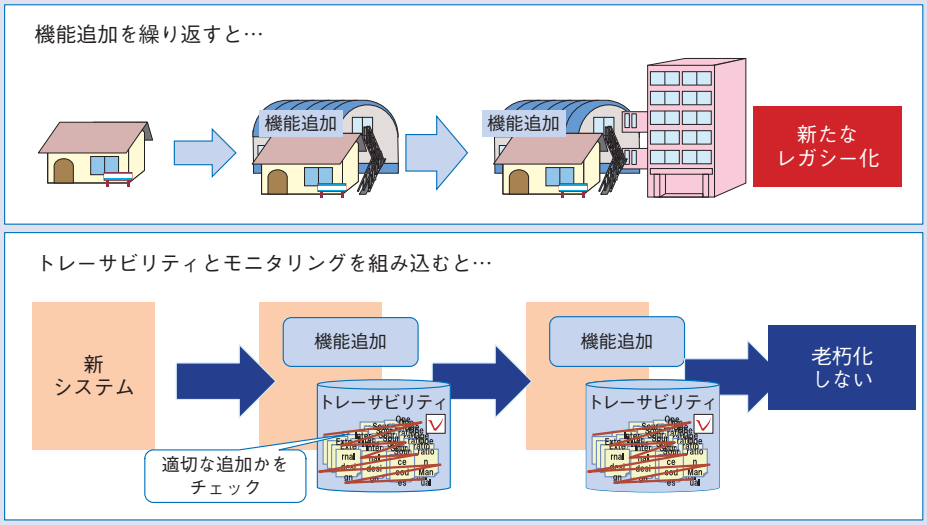


図 4　再レガシー化の防止

も最小限に防ぐことができます．

しかし，最新化されたシステムであっても，設計書がメンテナンスされずにソースコードと設計書が乖離すると，再びシステムはレガシー化します（**図 4**）．設計書の不備はシステムのブラックボックスや有識者不足を招き，システムの複雑化が加速して運用コストの上昇を招きます．小さなシステム変更であっても多くの稼働と時間が必要となってくるのです．システムを改造する際は，ルールに従って設計書を最新化することが重要ですが，限られた時間と関係者だけで実施することは困難な状況です．

対策は，設計書間の整合性のチェックやルールに従った変更が行われているかをコンピュータで自動的にチェックをすることです．システムを改造する際の設計書やプログラムを常にチェックして変更ができれば，上流工程での不具合検出にもつながり，QCD（Quality, Cost, Delivery）の改善にも大きく貢献します．このチェックプロセスは人が実施すると抜け漏れや未実施の原因となり，継続した活動にはなりません．人に頼らずコンピュータで自動的に実施することが重要です．

開発とチェックの自動化

メインフレームの多くはレガシーシステムとも呼ばれ，人員の流動化や新しい生産技術の導入の遅れを招いています．システム更改をしようとしても，大規模・複雑化・有識者不足によって難易度も高くなっており，大きな投資が必要になります．しかし，システム更改をしても，再び規模や複雑さ，有識者不足という問題にぶつかっては意味がありません．ある事例では，システムの保守を10年も続けると規模は1.5倍にも肥大化し，新商品やサービスを導入する際に必要な期間は当初の2～4倍にもなることもあるのです．この負のスパイラルを断ち切り，情報システムを継続的に利用するために，人手に頼らない開発とチェックの自動化に移行することがますます重要になります．

■参考文献
⑴ 我妻：“NTTデータにおけるソフトウェア開発自動化の取り組み,” NTT技術ジャーナル，Vol.26，No.10，pp.14-19，2014.
⑵ 金子：“真の上流工程へのシフトを実現するTERASOLUNA Simulator,” NTT技術ジャーナル，Vol.26，No.10，pp.20-23，2014.

谷野　英昭

レガシーシステムの問題は，今後はオープンシステムでも起きることが予想されるため，社会的な課題に発展することも考えられます．そのため，この問題に対してはさらに加速して取り組んでいきます．

◆問い合わせ先
NTTデータ
技術開発本部
ソフトウェア工学推進センタ
E-mail　case-nttgjournal@kits.nttdata.co.jp