Automation by Design

ITシステム運用の自動化を前提としたシステム設計のアプローチ

クラウド・ネイティブな設計で作られた最新のシステムは、クラウドの自動化機能やオーケストレーションによって高度に自動化されたシステム構築が期待されます。一方で、ITシステム全体の運用品質を向上させるには、従来型のクラウド化の対象とならない既存IT資産にも自動化の施策が必要になります。本稿では、システム運用の自動化を前提としてシステム運用の再設計を行うアプローチ「Automation by Design」の設計ポイントを設計例とともに解説します。

1. はじめに

クラウド・コンピューティングを基盤とした自律的なオーケストレーションなどの技術がオープンソースを中心に発展し、最近ではサーバーレス・アーキテクチャーによるクラウド・ネイティブなシステム構築が可能となりました。しかし、企業内には従来型アーキテクチャーのIT資産が依然として重要なビジネス・プロセスを担っています。従来型のシステムをクラウド・ネイティブな設計に変換することは、アプリケーションの大規模な改修が必要となり困難を伴うのが実情です。

従来型のシステムをクラウド環境に単純に移植しても、インフラを仮想化するだけでは運用は変わらず、システム運用におけるクラウド・ネイティブへの転換は進みません。こうした場合、クラウドであるかどうかにかかわらず従来型のサーバーOS単位のオペレーションが続いてしまいます。クラウド・ネイティブなアプリケーションでは、システム運用を自動化することで稼働の予測可能性を高めています。しかし、従来どおりのマニュアル対応が必要なシステムとの連携が残っていると、システム全体としてはマニュアル運用が残ってしまいます。“組織全体の部品の品質が高い水準で統一されている場合にパフォーマンスが最大となる”という「O-リングの理論」[1]に基づくと、マニュアル運用している既存システムの存在が、全体のシステム・パフォーマンスを阻害するということになります。

IBMは既存システムを再設計する際に、自動化のコンポーネントを最大限利用して自律的な運用システムを構築する「Automation by Design」というアプローチを用いています（**図1**）。人による手作業や柔軟な判断に基づく運用マニュアルを前提とした従来どおりの設計では、後から運用を自動化することが困難です。ハードウェアの老朽化による更新やソフトウェアのバージョンアップなどの避けられないシステム更新の際に、最初から完全に自動化して運用されることを前提とした再設計を行うことで、安定的で予測可能性の高い運用を実現し、システム全体として高いパフォーマンスを発揮する基盤を形成することができます。

2. Automation by Designの設計ポイント

Automation by Designにおけるシステム設計のポイントは以下の3つです。

- 自動化技術を採用したベースインフラ
- 非機能要件の変化に対応するオペレーションの自動化
- デリバリー・パイプラインとテスト

Automation by Designでは、まず自動運用に求められるベースインフラとして、自動化に対応する監視やバックアップなどの運用コンポーネント、KVMやESXi

のようなハイパーバイザーやハードウェアの制御プログラムの確保が必要となります。同時に、それらに対する管理ネットワークからの接続も必要です。ベースインフラが運用機能を用いてシステムの初期の非機能要件を満たすのに加えて、非機能要件の変化に対応するようにオペレーションの自動化を行います。ここで要件の変化に応じることができるインフラ機能と自動化の設計を行います。自動化を維持しながら継続的にオペレーションを行うには、リリースやテストなどを通じて必要な機能性を保証するデリバリー・パイプラインのプロセスも重要になります。

2-1.自動化技術を採用したベースインフラ

ベースインフラは、運用管理機能、運用プロセス機能、および自動化エンジンの3つのコンポーネントから成っています（**図2**）。

運用管理機能は、監視、システムの停止や起動、バックアップ、ジョブのスケジューリング、データの遠隔地保管や災害対策サイトへの切り替えといった運用上のユースケースに対応し、運用対象システムのプロビジョニング、消去・起動、停止などの操作やOS・ミドルウェア・アプリケーションなどの操作を行います。これらの機能はハイパーバイザー製品や監視・バックアップなどのソフトウェア製品やオープンソースを活用して構築します。

運用プロセス機能は、ワークフロー製品やルールベースでの判断エンジンを持つ監視製品などで構築し、ITILなどの運用プロセスを実現します。従来の運用プロセスは人間の作業を前提としていましたが、自動化したプロセスにより各作業間をつなぎ連続したプロセスを定義します。

自動化エンジンは、運用管理機能や対象システムに対する操作を人に代わって自動的に行う機能を提供し、運用プロセス機能の中で人と同じように活動します。監視およびインシデント対応など運用管理機能のインターフェースはさまざまです。あるクラウド・ベンダーはアラート情報をメールで発信しますが、ストレージの仮想アプライアンスはメールでアラート情報を送信するか監視サーバーにSNMPトラップを送信するか選択できます。アプリケーションやミドルウェアの稼働監視のために監視ソフトを導入することも多くあります。自動化エンジンが効率的に働くためには、運用プロセス機能で監視システムが受け取ったアラートやイベントの情報を集約しルールに基づいた高度な判断をする「Monitoring & Event Management※」が必要になります。対象システムが十分なAPIやログ、メッセージングなどのプログラム対応できるインターフェースがない場合、自動化エンジンとしてRPA（Robotic Process Automation）を用いることで、sshやPower Shellなどのコンソール、

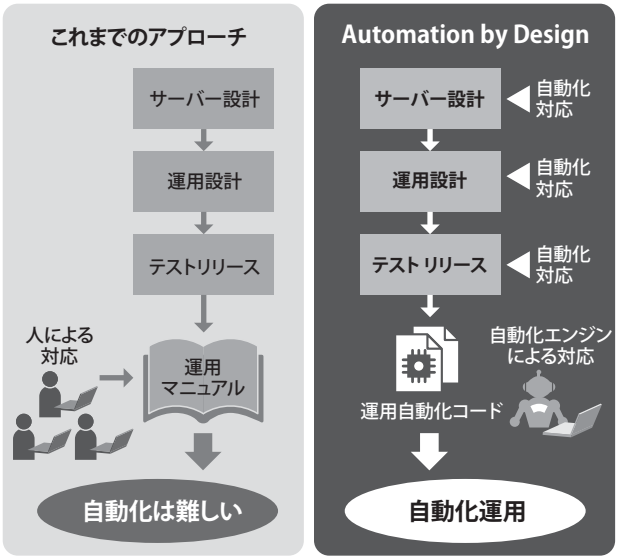

図1. Automation by Designにおける自動化アプローチ

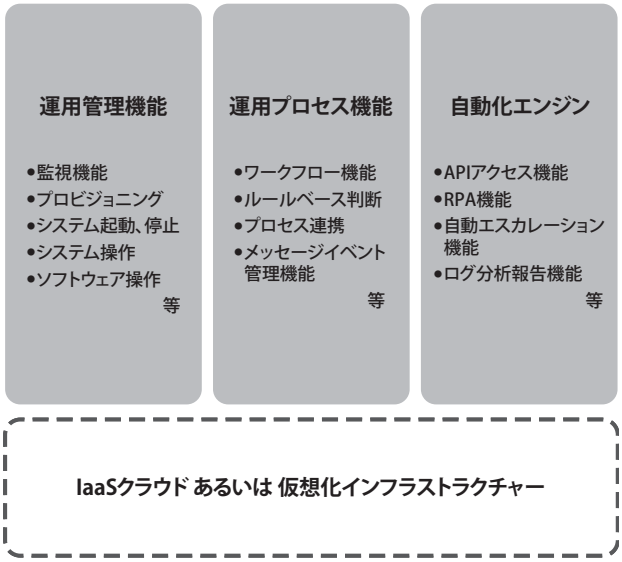

図2. Automation by Designのベースインフラ

Windowsやブラウザーの画面操作など、従来人間が行っていた操作をプログラム化し、自動化プロセスから呼び出すことを可能にします。

自動化エンジンは、イベントに対応したスクリプトを実行できるプラットフォームと、これまで人が対応してきたコマンドラインのようなユーザー・インターフェースをロボットが操作するRPAで構成されます。また、**図3**で示すように自動化エンジンは運用プロセス機能の中でまるで運用メンバーのように操作に参加し、これまで人が行っていたプロセスを代替します。人間の判断が必要な場合はインシデント管理プロセスの上で自動的にエスカレーションを行います。自動化エンジンはアラート対応が継続できない想定外の事象が発生した場合でもインシデントを適切な技術者の所属グループに転送し、そこまでに行った調査ログの必要部分を抜粋して以後の調査に有用な情報をインシデント・チケットに添付することができます。これまで通常のオペレーターが30分以上掛かっていたトラブル対応報告を、数分に短縮した事例もあります。

※Monitoring & Event Management：ITシステム運用を支援する「IBM NetCool Operations Insight」[2]を活用した「IBM Services Platform with Watson」の監視イベント管理コンポーネント

2-2.非機能要件に対応するオペレーションの自動化

Automation by Designでは、刻々と変わる非機能要件に対して、従来の運用担当者が細やかな配慮をして行ってきた運用手順の調整を必須の機能と考え、設計段階で自動化を組み込む際に変化への対応を考慮します。可用性を維持するためにはクラウドの単一障害点や、爆発的なユーザーの増加、データの伸びにも対応してシステムを健全に保たなくてはなりません。セキュリティーの管理においては、新たな脅威への対応や監査基準の変化にも対応が必要です。さらに開発時には明らかになっていなかった非機能要件が出現することも考慮しなくてはなりません。

IBMはこれまで、戦略的アウトソーシング・サービスで提供するシステム運用の品質と効率を高めるために、システム管理製品を統合し自動化ツールの開発投資を行ってきました。Automation by Designは、運用の自動化に必要なツール群が現実の運用品質を満足するレベルに到達してきたことを背景に、自動化された運用を前提にシステムを設計しテスト・リリースするというアプローチが可能になりました。

以下で、オペレーショナル・アスペクト、可用性管理、パフォーマンス・キャパシティー管理、セキュリティー

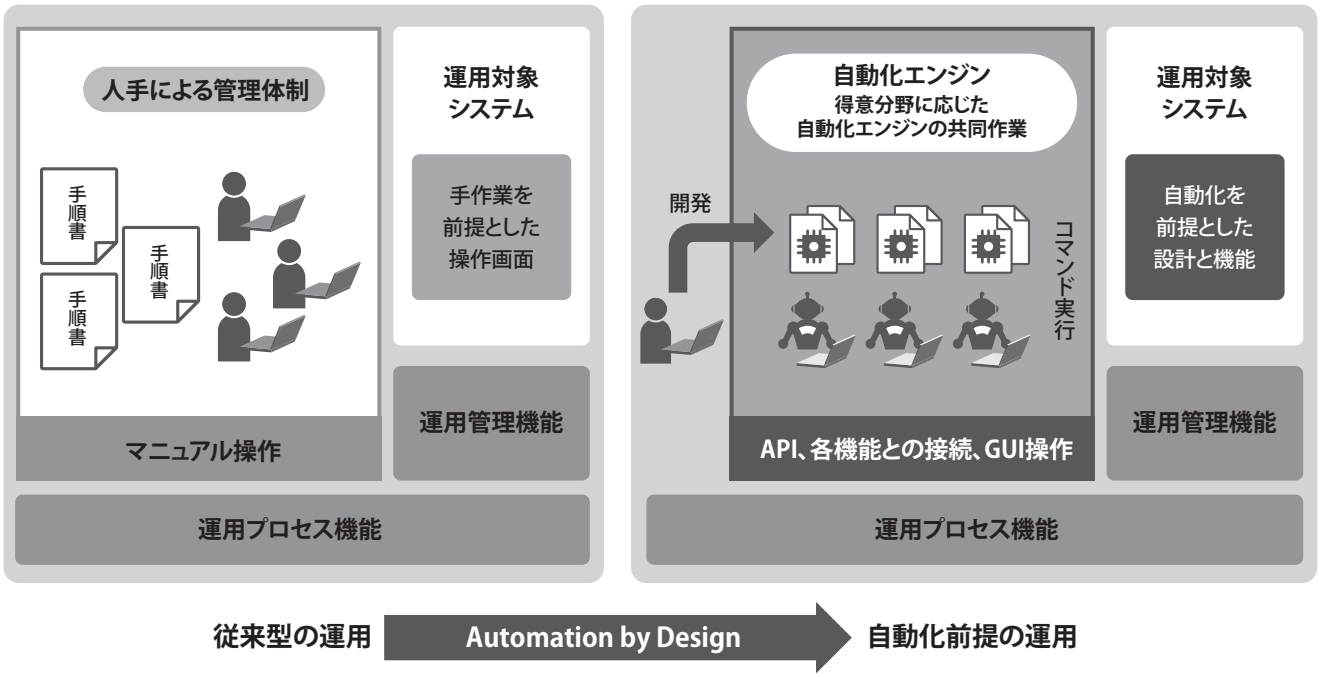

図3. 自動化を前提にする場合の運用の変化

というそれぞれの非機能要件領域で、自動化を前提とした設計する際のキーポイントを解説します。

●オペレーショナル・アスペクト

日常的な運用において自動化対応するべきことは数多くあります。これまで、例えばデータ・バックアップでは、オンラインのグレースフル・シャットダウン、オンライン・ユーザーの確認、停止点の確認、データ・バックアップの起動、バックアップ・データの確認と退避、オンライン再開、ユーザー・ログインの確認というような手順がマニュアル化されて手動で運用されてきました。Automation by Designでは、あらかじめこれらの手順が自動化されたログ・イベントベースで実行できる自動化エンジンを前提に設計します。こうしておくことで、データ・バックアップの環境が変更された場合でも、プログラムを変更するだけで柔軟に対応できる自動化対応が可能になります。これまで暗黙的に対応してきた例外事象もルールベースに記述しテストによって確認することで確実な自動化を実現します。手作業前提でリリースしてしまったサーバーの操作を後から自動化することは例外処理が多く困難を伴いますが、自動化を前提にテスト、リリースできれば多くの作業が自動処理できるようになります。

●可用性管理

自動化対応でもっとも大きく変化するのが可用性管理です。これまで固定的なインフラにおいて高い可用性を求められるシステムには代替機がスタンバイするクラスターが用いられてきました。しかし、二重化のコストは高く接続も複雑になるため、どのシステムでも採用するというわけにはいきませんでした。一方、クラウドや自動化をベースにしたシステムでは、サーバー障害に対してクラスタリングだけではなく、プロビジョニング、ビルド、リリースというように、壊れたところを修復する自律的な修復プロセスを自動化することが可能になります。従来型のサーバーであってもクラウド上のビルド・テストが自動的に行える「Infrastructure as Code」(技術解説「Infrastructure as CodeとServer Lifecycle Automationの活用」49ページ参照)が実現できることでMTTR(Mean Time To Recovery：平均修復時間)を劇的に下げられます。また、クラウド環境ではデプロイ先に異なるAvailability Zoneを選択することもできるので、クラウド障害にも自動化で対応することが可能になります。こうしたことも設計段階で自動化を考慮していなくてはデプロイメントのインフラストラクチャーを構築することができず、従来型のクラスターを前提にせざるを得ません。

●パフォーマンス・キャパシティー管理

サーバー・プラットフォームのパフォーマンスやキャパシティー設計では、想定した非機能要件から数学的手法や経験的数値を利用して初期値と変動要素を設計します。オンライン・ユーザーの増加に対応したヒープサイズの調整、vCPUの割付調整、並列化している場合には負荷分散装置と協働したオートスケールの調整など対応策がとられます。ストレージではリニアに増加するデータとオフラインに退避するデータのバランスを取ったり、緊急時にログなどの運用データ領域をアプリケーションに明け渡したりするような調整が想定されます。Automation by Designではこれらの判断基準をルールエンジン化して対応します。想定される調整要求に対して適切な手法手段を選択し、ロードバランサーやストレージ制御など周辺環境と同期をとってプログラム実行できるように設計することが必要です。これも自動化を前提に設計することで、想定外の事態が発生した場合でもルールエンジンを調整することで再発を防止し、正確で洩れのない操作が期待できます。また、こうした経験的な調整スキルはこれまで属人化しやすく散逸しがちでしたが、ルールエンジンのコーディングに残されることで、知識を継承し資産化できるようになります。

●セキュリティー

セキュリティーの自動運用ではOSやミドルウェアのコンプライアンス、セキュリティー・ソフトウェアの運用と監視などが対象となります。IBMのセキュリティーの自動化エンジンである「Continuous Compliance」は、コンプライアンス状況を常に監視して修正する冪等性(べきとうせい)管理を行っています[3]。こうしたツー

ルを活用することで、セキュリティー・コンプライアンスを一元管理することができ、例えばシステム・ファイルの権限の基本ルールを変更するようなコンプライアンス条件の変化にもセンター側だけで対応できるようになります。また、セキュリティー・ソフトウェアの管理も自動化できます。これまでのセキュリティー・ソフトウェアへのポリシー配布は、配布までは自動化されていても、配布後の確認やサーバーへの影響調査などは人手に頼っていました。セキュリティー・ソフトの管理を自動化前提で設計することで配布や配布後の確認など定期的に手間のかかっていた作業を機械化することができます。サーバー・セキュリティーのデプロイメント・プロセスを自動化しておくことで、セキュリティー・インシデントに対する回復力を高める設計が可能になります。

複合的なソリューションとして自動化を前提に実現した遠隔地切り替えでは、これまでは年に一度の災害対策リハーサル試験を行っていた運用プロセスを、週や月に1回の切替えを定例化して日常オペレーションとすることができます。自動化を前提に日常的に切り替えができれば、セキュリティー脆弱性への対応パッチを待機系に適用して切り替えるといった2センターの効率的なオペレーションも可能になります。また、片側だけに変更を適用し忘れるといった期間をあけることによる想定外の差分発生を防止できます。

2-3.デリバリー・パイプラインとテスト

自動化インフラ上でシステムを運用するには、継続的インテグレーション（以下、CI：Continuous Integration）を採用したデリバリー・パイプラインでのテストの仕掛けが非常に重要です。そのパイプライン上では運用で発生したインシデントに対応しながら継続的にオペレーションを改善していくことが可能になります。

システム構築とデプロイ作業は、従来の導入手順書や操作手順書という成果物を作成して引き継ぎを行う作業から、デリバリー・パイプライン上で自動的に動作する「テストケース」「テストコード」「テストスタブやモック」「CIツール（Jenkins/Travis）でのテストログ」「提供するAPIのリファレンスやサンプル実装」と、それらを保管する「ソースコード・レポジトリー」に置き換わります。

デプロイメントで発生するテスト環境の作り直しやグリーン・ブルー・デプロイメントのための待機環境の構築、カナリヤ・デプロイメントの増分サーバー追加といった目的別の運用ユースケースを「IBM UrbanCode Deploy」[4]のような自動化されたデプロイメント・ツール上に実現することで、再帰的にテストすることが可能になります。例えば、障害発生時のインシデント管理プロセスを自動化した上で擬似障害の発生から回復の確認までの一連の動作テストを実現できるようになります。

デリバリー・パイプラインの目的は、単にシステムを

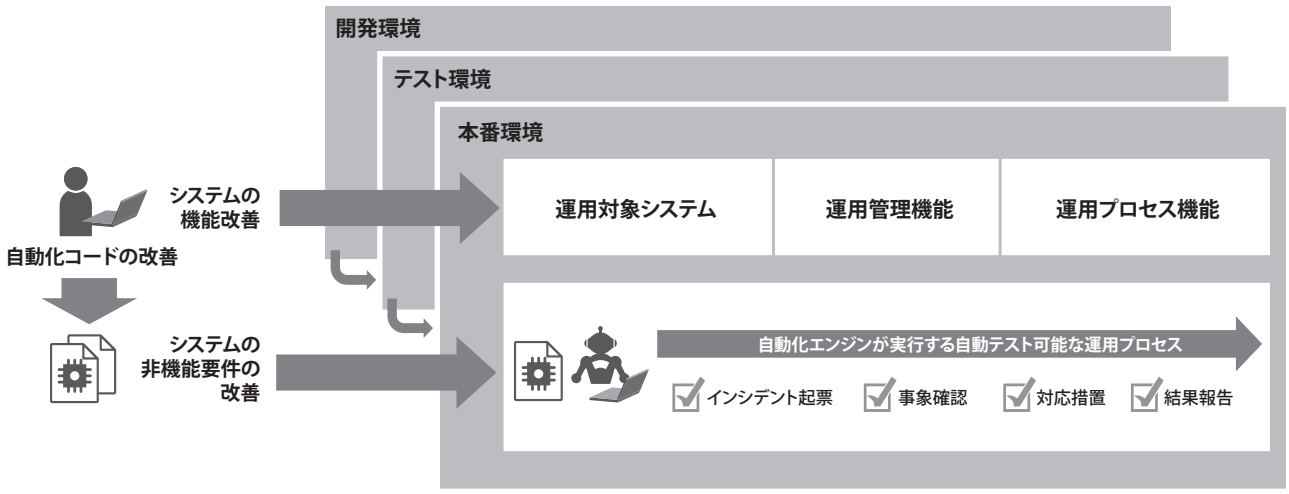

図4. デリバリー・パイプラインによる機能改善と非機能要件の改善

デプロイし動かすことではなく、運用サイドで自動化機能のCIを実現し、非機能要件を改善することです。CIでは**図4**の上段に示すようにパイプラインを設計し機能の改善を繰り返し行います。Automation by Designでは自動化した運用プロセスについても自動化コードの改定をパイプライン上で行い非機能要件を改善していきます。システム上のセキュリティー異常やパフォーマンスの異常検知に対する自動化対応に不足や不備があれば、自動化プログラムを作成・修正してパイプラインからリリースし、その繰り返しによって継続的にシステムの自動化による品質を改善できるようになります。一連の自動化処理の不整合があった場合に、課題点をインシデントとして自動的に起票し技術者に報告するプロセスも必要で、それによって自動化が未実現の手作業は徐々にバックログとなります。これまでインシデントの発生原因を管理していた問題管理プロセスは、発生した問題の根本原因対応に加え、自動対処内容を改善するための自動対応コード上の機能改善やバグを修復するというソフトウェア的な手法に代わることになります。

3. おわりに

Automation by Designのアプローチにより、自動化を前提として運用品質を改善する設計方法を説明しました。また、その際に必要となる運用全体のアーキテクチャーと継続的な改善のための運用プロセスを解説しました。

重要なシステムの運用継続と品質改善は避けられません。新たなセキュリティー脅威への対応や使用ソフトウェアのサポートの確保のためにシステムの更新は必要になります。いつかは必ず訪れるシステム老朽化の対策時にAutomation by Designのアプローチを用いることで、運用テストコードによる動作確認により客観的でリスクの少ない状態を確保できます。ひとたびそのような運用サイクルが始まれば、その後のシステム更新においても最小限の負担で価値のあるシステムを維持できます。

人間が行う運用作業は多岐にわたります。主要なプロセスを自動化したとしても新しいニーズへの対応をソフトウェアで実現するには時間が掛かります。一方、固有のプロセスの自動化に対して属人化した判断を紐解き、ソフトウェアによる判断を可能にするには事前のパターン分類が必要となりますが、これらの作業には時間が掛かるため優先順位を高くする必要があります。特に、不要な作業を見極めて何かを止める決断は人間にしかできません。「IBM Services Platform with Watson」(技術解説「IBM Services Platform with Watson」44ページ参照)では、コグニティブ技術を用いたアプローチでこれらの判断を支援しますが、プロセスの改善自体は人間にしかできないのが現状です。自動化の次のステップは、機械学習により自動化エンジンが新しい作業プロセスを提案し、デリバリー・パイプライン上でその作業プロセスをシミュレーション可能にすることかもしれません。

自動化技術の発展は今後も継続しシステム設計の基準を変えていきます。本稿が既存システムに対して常に最新の自動化技術を装備させる際の方針や具体例を示し、さらに高い運用品質を実現するきっかけになることを期待します。

[参考文献]
[1] Kremer, Michael: "The O-Ring Theory of Economic Development". The Quarterly Journal of Economics. Oxford University Press. 108 (3): 551–575. doi:10.2307/2118400. JSTOR 2118400 (1993)
[2] IBM : IBM Netcool Operations Insight, https://www.ibm.com/jp-ja/marketplace/it-operations-management
[3] IBM: IBM Services Platform with Watson, Continuous Compliance, https://www.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=IXV12346USEN
[4] IBM : IBM UrbanCode Deploy , https://www.ibm.com/jp-ja/marketplace/application-release-automation

日本アイ・ビー・エム株式会社
グローバル・テクノロジー・サービス事業
アドバンスド・オートメーション
シニア・アーキテクト

南波 邦雄
Kunio Namba

1999年日本IBM入社。戦略的アウトソーシング・サービスにて、金融系や製造業のお客様のシステム運用の自動化を担当。自らが設計する自動対応システムに自動化を適用し、そのノウハウを、お客様のシステムの自動化に適用することで自動化システム自体の品質改善とお客様システムの運用品質の両方を改善中。