RED HAT®
CONSULTING

redhat.

ホワイトペーパー

ビジネス自動化のための構造化設計手法

ソフトウェア開発のインフラストラクチャと手法の進歩により、ビジネス自動化をモダナイズするためのツールや機会を活用できるようになりました。最新ソフトウェア開発のベストプラクティスをビジネス自動化へ適用する設計手法は、リスクを上げることなく厳格なエンジニアリング・プラクティスを組織に導入するために役立ちます。

エグゼクティブサマリー

多くの大手企業において、 IT 部門は競争に対応し、業界の規制を順守し、顧客の関心を引き付けるためのツールを提供する必要があります。IT 部門は、迅速かつ頻繁な変更に対応可能な柔軟性を備えた高品質のソリューションを、予測可能なコスト、できれば低コストで提供することが期待されています。

ビジネス上のステークホルダーに必要なソリューションを必要なタイミングで提供する 2 つの方法が、カスタムソフトウェア開発とビジネス自動化です。しかし、この 2 つのアプローチにはそれぞれ利点と欠点があります。ソフトウェア開発は、厳密なプロセスを用いたエンジニアリング手法ですが、時間がかかることで知られています。ビジネス自動化の場合、技術者以外のステークホルダーがビジネスロジックを組み込めるため、市場投入時間を短縮できますが、リスクは増大します。

ソフトウェア開発のインフラストラクチャと手法の進歩により、ビジネス自動化をモダナイズするためのツールや機会を活用できるようになりました。最新ソフトウェア開発のベストプラクティスをビジネス自動化へ適用する設計手法は、組織がこの機会を活用し、リスクを上げることなく厳格なエンジニアリング・プラクティスを導入するために役立ちます。この手法をビジネス自動化に利用することで、企業は専門家の協力のもとビジネスロジックをコード化し、品質の向上、市場投入期間の短縮、および最新ソフトウェア開発コストの予測と低減を実現することができます。

ビジネス自動化とソフトウェア開発の進化

IT リーダーは常にビジネス要求を満たすための新しい製品やテクノロジーを探しています。しかし、商用ソフトウェアでは迅速なイノベーションや十分な差別化を実現できない場合が多く、企業の IT チームがカスタムソフトウェアを開発する必要が生じます。

従来、ソフトウェア開発は時間がかかる上に複雑で、初期段階からビジネスの専門スタッフと開発者の間で効果的なコミュニケーションが必要とされました。一般的には、ウォーターフォール型の手法が採用されてきました。この手法では、ビジネスの専門スタッフが事前に開発者に要件を提供し、それに基づいて開発者がアプリケーションを設計、開発、テストします。しかし、ビジネスの専門スタッフがアプリケーションを評価できるのは完成後になるため、変更作業は困難でコストもかかりました。

ビジネス自動化は、ビジネスプロセスの自動化とも呼ばれ、IT 部門主導のソフトウェア開発に代わる方法として登場しました。この方法では、ビジネスの専門スタッフが、必要なビジネスロジックを迅速かつ反復的に定義して実行できます。ビジネスロジックは、アプリケーションの機能としてではなく、プロセス、ルール、またはワークフローのセットとして表現されます。これらのルールは、特定の条件やイベントに応答して、エンジンによって実行されます。

たとえば、金融機関の新規顧客の登録プロセスは、マネーロンダリングやその他の違法行為を防止するため政府によってあらゆるレベルで規制されていますが、これらの規制は頻繁に変更されます。金融機関のコンプライアンス担当者は、ビジネス自動化を使用して、新しい規制の条件や関連ルールを定義できます。そしてその後、ソフトウェア・アーキテクトや開発者などの IT スタッフの助けを借りずに、これらの新しいルールをエンジンにデプロイすることができます。

ビジネス自動化とカスタムソフトウェア開発には、ソフトウェアシステムにビジネスロジックを組み込むという共通の目的がありますが、それぞれが別個に確立されていました。その結果、IT の孤立化と断片化が生じ、サーバー、オペレーティングシステム、およびその他のソフトウェア・ライセンスの重複や、保守とサポートなどによりコストが増大していました。また、ソフトウェア開発の制約を回避することで、厳密なエンジニアリング・プラクティスを回避することになり、ビジネス自動化によって意図せずリスクが高まってしまう可能性もありました。たとえば、バージョン管理の変更履歴の保守、テスト済みで本番リリースの準備ができたものの追跡、構成 (多くのコンポーネントのライフサイクルを含む) の管理といったソフトウェア・エンジニアリングの原則は、ビジネス自動化環境では見過ごされる場合がありました。

この数十年間で、ソフトウェアの設計と開発は大きく変化しました。多くの場合、これらの変化は、最初にビジネス自動化の台頭を招いたのと同じ批判に対応するためのものでした。ソフトウェア・アーキテクチャは、ビルドに時間がかかり変更も難しい、巨大でモノリシックなアプリケーションをベースにした設計から、単一の目的のために設計され、作成、更新、保守が容易なアプリケーション・コンポーネントによる、サービスに基づいたアーキテクチャへと変化しました。

ソフトウェア開発手法も変化しました。従来のウォーターフォール型の手法は慎重な計画と明確な段階設定が特徴であり、プロセスの一部を繰り返すことは困難かつ複雑です。この方法は、反復型のアプローチに置き換わりました。つまり、アプリケーションの迅速な実行、ステークホルダーによる確認、頻繁な繰り返しを伴う変更を行うことを意図した手法です。エクストリーム・プログラミング、スクラム、大規模アジャイル・フレームワーク (SAFe) といったこれらのフレームワークは、技術の進歩に伴って徐々に改善されてきました。現在では、継続的インテグレーションおよび継続的デリバリー (CI/CD) のプラクティスが、デプロイメントの迅速かつ反復的なアプローチを拡張しています。これらのプラクティスは、毎日複数回コードを組み込み、開発中に何度もテストし、ソフトウェアをいつでもリリースできるようにするという考えに基づいています。

ビジネス自動化のための最新の設計手法

多くのビジネス自動化ソリューションは、ソフトウェアの設計と開発の進歩を活用する機会を逸しています。最新のソフトウェア開発のメリットをビジネス自動化にもたらす効果的な設計手法では、設計上の懸案事項を一連のアーキテクチャ・コンポーネントにまとめます。コンポーネントごとに、実装に関する重要な意思決定を少なくとも 1 つ行います。この手法で構築された環境において、組織は既存のソフトウェア開発インフラストラクチャをビジネス自動化に適用し、両方の環境をサポートできます。

アーキテクチャ・コンポーネントは、以下に示す効果的なビジネス自動化手順の主要な活動に対応します。

1. **モデル化：**アプリケーション・コードの開発またはビジネス自動化の基本要素 (ルール、イベント、プロセス、ワークフロー) の定義によって、ビジネスロジックをモデル化し、それを定義するシナリオをテストします。
2. **バージョン化：**監査可能な変更履歴を含む、モデルの成果物をバージョン管理します。
3. **ビルド：**成果物のコレクションからデプロイ可能なパッケージをビルドし、テスト自動化を実行し、パッケージの履歴を保守します。
4. **保存：**パッケージをマスターリポジトリに保存します。

5. **デプロイ：**実行環境 (コードの場合はアプリケーションサーバー、ビジネス自動化要素の場合はエンジンなど) にパッケージをデプロイしてテストします。
6. **実行：**ビジネスロジックを実行します。

以下のセクションでは、アーキテクチャ・コンポーネントと、重要な実装の意思決定について説明します。環境の複雑さ、機能、コストは、各コンポーネントに関する意思決定に依存します。

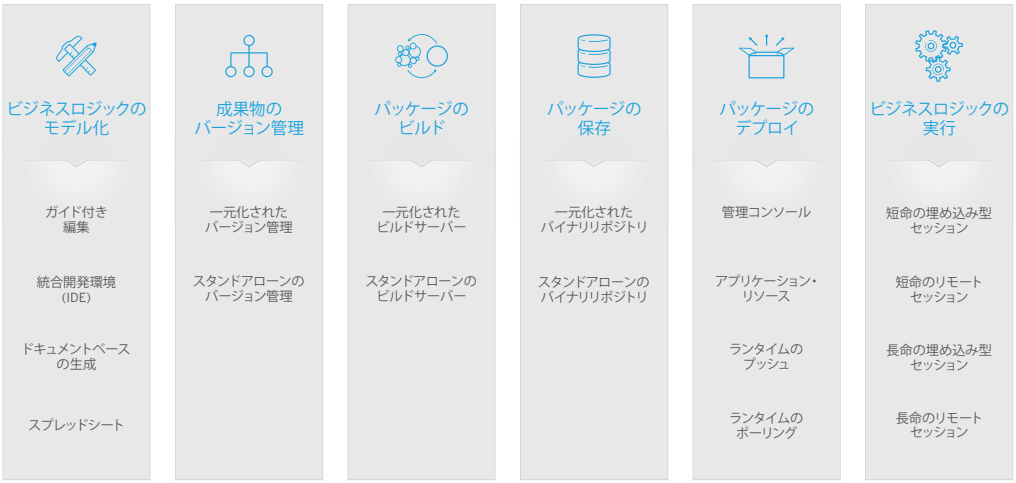

図 1. ビジネス自動化のためのアーキテクチャ・コンポーネントとその実装に関する意思決定

1. モデル化

モデル化は、実行するビジネスロジックを捉えるタスクです。ユーザーは通常、期待するシステムの動作に関する自分の知識に基づいてビジネスロジックをモデル化します。ビジネスロジックは、アプリケーション・コード、ビジネス自動化の要素、または両方の組み合わせによってモデル化できます。

アプリケーション・コードでは、プログラマーはプログラミング言語を使用してビジネスロジックをモデル化します。多くのプログラマーは統合開発環境 (IDE) を使用してコードを記述しますが、標準準拠のコードを作成するものである限り、任意のテキストエディターを使用できます。

ビジネス自動化では、ビジネス分野の専門家は以下のいくつかの方法でルール、プロセス、イベント、およびワークフローを作成することにより、ビジネスロジックをモデル化します。

- **ガイド付き編集ツール：**ガイド付き編集ツールでは、技術的な詳細情報は表示されず、技術的な知識を必要としない直感的なインタフェースを提供します。このツールを使用して、ユーザーはテストケースを定義し、精度を検証します。ガイド付き編集は、ビジネスの専門スタッフがプログラマーの助けなしにモデル化を実行するためのツールを提供し、ビジネス自動化の本来の目的を実現します。
- **スプレッドシート：**ビジネスの専門スタッフは、スプレッドシートを使用して、ルールをデシジョンテーブルとして定義することもできます。これは一般に、行全体で繰り返しパターンを使って作成されます。大抵の場合、ビジネスユーザーはスプレッドシートを使い慣れているため、各行の値を簡単に編集し、目的のロジックを定義することができます。スプレッドシートは、プロセスのモデル化にも使用できます。

- **ドキュメントベースの生成：**このアプローチでは、テンプレートでルールまたはプロセスの構造を定義します。契約書やプロモーション・オファーなど、ドキュメントに格納されているビジネスデータがテンプレートに適用され、ルールやプロセスが生成されます。
- **統合開発環境 (IDE)：**IDE は、プログラマーがアプリケーション・コードを記述する際に使用されますが、ビジネス自動化要素の作成に使用することもできます。IDE では、ユーザーがビジネスルールとプロセス図を作成するためにプラグインが必要です。IDE では構文のハイライトなどの追加機能を使用でき、技術に詳しいユーザーにとって使い慣れた作業環境が提供されます。

ビジネスロジックをモデル化する際、ビジネスの専門スタッフはビヘイビア駆動開発 (BDD) を使用して、アプリケーションの開発および最終テストの指針を提供できます。BDD では、ビジネスの専門スタッフが、アプリケーションに期待される動作を理解しやすい言語で記述した短いシナリオを作成し、それがアプリケーションの要件となります。開発者 (あるいは、ビジネス自動化の手法ではビジネスの専門スタッフ) が、これらのシナリオを使ってビジネスロジックをモデル化します。開発ライフサイクルの後の段階においても、これらと同じシナリオが、アプリケーションの自動テストの基礎として使用されます。

2. バージョン化

バージョン管理システムは、ビジネス・ロジック成果物のマスターリポジトリです。ビジネスロジックをアプリケーション・コードまたはビジネスルールのいずれとして捉える場合も、使用するツールに関係なく、作成された成果物を保存する必要があります。しかし、バージョン管理とは、随時加えられたすべての変更を記録するために成果物を保存し、各変更の理由を記録し、特定の過去のバージョンの回復を可能にするだけのことではありません。バージョン管理は成果物の複数の分岐に対応し、複数の変更シーケンスを追跡することもできます。

従来のビジネス自動化製品では、データベースを使用してビジネス自動化の成果物のバージョンを保存しますが、データベースにはバージョン管理ツールが提供するメリットのいくつかが欠けています。アプリケーション・コードには、Git、Mercurial、Apache Subversion などの標準のファイルベースのバージョン管理システムが広く使用されています。多くの組織ではすでにこれらのシステムを使用して、一元化された共有バージョン管理を作成しています。既存のバージョン管理システムを使用することで、プログラマーやビジネスの専門スタッフは、セキュリティ、データの保持やバックアップなどを心配する必要がなくなります。ただし、一元化されたバージョン管理インフラストラクチャが利用できない場合や複雑すぎる場合、または統合にコストがかかる場合は、ビジネスロジック成果物をスタンドアローンの専用バージョン管理システムに保存することができます。

3. ビルド

ビルドシステムは、最終的なリリースのために成果物を準備し、パッケージとして組み合わせるための主要なプロセスです。理想的にはユーザーの介入をほとんどあるいはまったく必要とせず、効率的な一貫した方法で成果物をパッケージとしてビルドすることが重要です。

パッケージは通常、Apache Maven などのビルド自動化ツールで作成されます。ビルド自動化ツールは、バージョン管理システムから成果物を取得するために必要なコマンドを実行するだけでなく、最終的なリリースのためにそれらを準備して組み合わせるように設定されています。ビルドを自動化することで、新しい成果物がバージョン管理システムに保存されるたびに新しいパッケージが作成されるようにできます。

新しいパッケージの自動テストは、このステップで継続的インテグレーションの一環として完了します。このテストでは、統合エラーをすぐに特定し、追加コードがバージョン管理されて新しいパッケージがビルドされる前にエラーを解決できるようにします。その結果、統合エラーのあるパッケージはリリースプロセスのこの段階の先には進みません。

バージョン管理システムと同様に、多くの組織には既存のビルド自動化システムがあります。既存のシステムを使用することにより、ユーザーは新たなシステムのインストール、構成、および保守を行う必要がなくなります。さらに、多くの場合、パッケージ化されたソフトウェア資産は企業ポリシーおよび外部規則に準拠している必要がありますが、それらは一元化されたビルドシステムで既に対処済みです。

一元化されたビルドシステムが Maven をサポートしていない場合や、既存のシステムを使用する上で他に障壁がある場合は、別のビルドシステムを使用できます。別のビルドシステムを作成すると、複数のビルドシステム (たとえば、アプリケーション・チームごとに 1 つ、またはプログラマーやビジネス専門家ごとに 1 つ) が存在することになります。

4. 保存

パッケージリポジトリは、すぐにデプロイ可能なソフトウェア・パッケージのマスターソースです。ソフトウェア・パッケージは、異なるデータセンターの異なるサーバーに複数回デプロイされることがよくあります。保存されているパッケージをリポジトリからデプロイする方法は、デプロイの必要が生じるたびにパッケージを再構築するよりも効率的で一貫性があり、毎回まったく同じパッケージを確実にデプロイできます。

前述のとおり、一元化されたリポジトリは、セキュリティ、ガバナンス、法令順守、データ保持、監査の要件に対処するために多く使用されます。さらに、スタンドアローンの Maven 準拠のリポジトリを使用することもできますが、その場合、結果として組織全体で複数のリポジトリを使用することになります。

5. デプロイ

ソフトウェア・パッケージは、パッケージリポジトリから、パッケージのタイプに応じた実行環境にデプロイされます。たとえば、スタンドアローンのバイナリ実行可能アプリケーションは、実行先のサーバーにコピーされます。一方、エンタープライズ Java アプリケーションはアプリケーション・サーバーにロードされます。ビジネス自動化パッケージは、ルールとプロセスの実行環境であるビジネス自動化エンジンにデプロイされます。

パッケージは、ユーザーの介入をほとんどあるいはまったく必要とせずにデプロイできます。ユーザーがデプロイを開始した場合でも、エラーの可能性を低減するため、デプロイのステップは事前定義され、事前に検証されます。デプロイの優先順位やスケジュールなど、デプロイメントルールも設定できます。

デプロイメントルールは、必要なリリース前テストの定義にも使用されます。パッケージをプロダクション環境にデプロイする前に、通常はパッケージを品質保証 (QA) 層とプリプロダクション層にデプロイしてテストします。これらのテストは、ビルドステップの自動テストよりも広範囲に行われます。たとえば、QA 層ではテスト担当者がユーザーとして操作して新しい機能を試し、プリプロダクション層では負荷テストとパフォーマンステストがしばしば行われます。これらの層でのテストが正常に完了した場合にのみ、パッケージはプロダクションにデプロイされます。

パッケージをデプロイする上で望ましい方法は、Push 型のデプロイです。この方法では、あるシステム上の自動化されたプロセスが、パッケージを実行環境にデプロイします。継続的デリバリー (CD) の手法では、パッケージがリポジトリ内で更新された際にデプロイメントを実行するか、スケジュールに従って実行するように設定できます。すべての実行環境には、デプロイメントルールに従って最新バージョンのパッケージがデプロイされ、ユーザーの介入は不要です。

同様の手法では、管理コンソール (通常はユーザー・インタフェースを備えた Web アプリケーション) を使用してパッケージをデプロイします。ユーザーは、パッケージをデプロイするためのパラメータ (名前、バージョン、ターゲット実行環境など) を設定し、手動でデプロイメントを開始します。その後、管理コンソールはパッケージを実行環境に push します。管理コンソールの push は自動ではなく、ユーザーが行う push です。

Pull 型 (ポーリング) の手法も使用できます。この手法では、実行環境が、更新されたパッケージを検索してリポジトリから取得します。この手法では、ビルドとデプロイのステップがより緊密に結びつき、運用は複雑になります。これは、更新されたパッケージがビルドされるとすぐに pull されるためで

す。たとえば、開発環境、テスト環境、プロダクション環境など複数の実行環境で同じリポジトリを使用すると、すべての環境が最新のビルドを pull します。デプロイを行う方法はいくつかありますが、Push 型の機構が好まれています。

ビジネス自動化の成果物の場合、別のデプロイ方法として、アプリケーション・リソース・アプローチがあります。この手法では、成果物が実行エンジンとともにアプリケーションにバンドルされます。これはシンプルで統一されたアプローチですが、モノリシックなアプリケーションが作成されます。デプロイメントは、ビジネス自動化パッケージというより、従来のアプリケーションに似たものとなります。このアプローチは、Push 型の手法または管理コンソールと併用することが理想的です。

6. 実行

一般に、実行環境には、ビジネスロジックを実行するために必要なすべてのサーバー、ストレージ、オペレーティングシステム、ネットワーク、その他のシステムが含まれますが、このホワイトペーパーでは、ビジネス自動化コンポーネントのみを取り上げます。

実行環境の実装上の考慮事項は、本紙で説明したアプローチの他のアーキテクチャ・コンポーネントの場合と異なります。モデル化のステップはユーザーのスキルや好みによって異なりますが、バージョン化、ビルド、保存のステップに必要なのは、リポジトリの場所を決定することだけです。また、決定した場所はいつでも変更できます。デプロイのステップに関する考慮事項は、パッケージが実行環境にどのように配置されるかに影響しますが、実際の実行方法には影響しません。

実行環境はビジネスロジックの実行方法に影響し、技術設計を通じてパフォーマンス、スケーラビリティ、信頼性に対応します。この環境における設計上の意思決定は、顧客満足度を得るための鍵や、成功と失敗を分かつ要素となる場合があります。

ビジネス自動化エンジンをアプリケーションの一部として配布するか、それとも一元化された共有サービスとして確立するかという決定は、設計上の重要な意思決定となります。組み込み実行の場合、エンジンはアプリケーションの一部となり、そのアプリケーション専用に機能します。アプリケーションとエンジンは、同じ Java™ 仮想マシンで一緒に実行され、最高のパフォーマンスを実現します。しかし、このタイプの実行では、エンジンが稼働しているすべての場所で追加のコンピューティング処理が必要になるため、コスト効率が低下します。

一方、一元化された共有サービスを確立すると、アプリケーション・ロジックは、ビジネス自動化エンジンや、それにデプロイされたルールやプロセスから分離されます。すると、一貫性のある単一のエンジンを、組織全体であらゆるアプリケーションに使用できます。その結果、アプリケーションで必要なコンピューティング処理が減り、必要に応じてこの共有エンジンをスケーリングすることで多くのアプリケーションをサポートできます。この手法を使用するには分散プログラミングの技術が必要ですが、さまざまなプロトコルを使用してエンジンにアクセスできるため、Java ベースではないアプリケーションでもエンジンを使用できます。

もう 1 つの重要な意思決定は、アプリケーションとエンジンのインタラクションの方法についてです。アプリケーションによっては、ルールを実行し、結果を 1 回のインタラクションで返します。こうした短命なセッションの場合、エンジンはルールの実行後にアプリケーション・データを保存しません。通常、アプリケーションは独立した永続ストレージ内のデータを管理し、必要に応じてエンジンに渡します。

また、数日、数週間、あるいはそれ以上にわたるビジネスプロセスを実行するアプリケーションもあります。こうしたインタラクションでは、アプリケーションはエンジンとの長命のセッションを作成し、エンジンは後続のステップで使用するためにアプリケーション・データを保存する役割を担います。この場合、他のアプリケーションがデータにアクセスすることもできます。

セッションの寿命と状態の管理は、実行環境のパフォーマンスとスケーラビリティに重大な影響を及ぼす可能性があります。短命セッションは複雑さを加えることなく水平方向にスケーリングできます。一方、長命のセッションは垂直方向または水平方向にスケーリングできますが、適切に設計されたデータベースが必要です。一般的に、リソース要件のため、長命セッションは必要な場合 (ビジネスプロセス管理など) にのみ使用されます。

RED HAT®
CONSULTING

redhat.

まとめ

ビジネス自動化は、ソフトウェア開発では現代のビジネスニーズを満たすことができなくなったために登場しました。しかし、これらのニーズを満たすために、ビジネス自動化はリスクの増大やその他の課題も生み出しました。この 20 年間で、イノベーションのスピードが加速し、従来のビジネス自動化ソリューションでは対応しきれなくなりました。今や経済は、成熟した市場の破壊的デジタル革新と、急速に進化し複雑化する規制環境によって定義されています。

Puppet による 2015 年の「State of DevOps Report」[1] などの業界調査によれば、好業績のソフトウェア企業はこの激動の中で競争力を維持するための準備を着々と進めています。したがって、ビジネス自動化ソリューションに、ソフトウェア開発とデプロイメントのための最新のインフラストラクチャと手法を組み込むことが必要になっています。

Red Hat® コンサルティングが提案するビジネス自動化 (Business Automation) プラクティスでは、このホワイトペーパーで紹介した統一的なアプローチを採用しています。最新のソフトウェア開発手法を回避するより、むしろ活用することで、ビジネス自動化の価値を実現できるよう企業をサポートします。

Red Hat コンサルティングのビジネス自動化プラクティスの詳細については、redhat.com/ja/resources/red-hat-consulting-discovery-session-business-automation にあるデータシートをご参照ください。

1 https://puppet.com/resources/white-paper/2015-state-of-devops-report

RED HAT について

オープンソースソリューションのプロバイダーとして世界をリードする Red Hat は、コミュニティとの協業により高い信頼性と性能を備えるクラウド、Linux、ミドルウェア、ストレージおよび仮想化テクノロジーを提供、さらにサポート、トレーニング、コンサルティングサービスも提供しています。Red Hat は、お客様、パートナーおよびオープンソースコミュニティのグローバルネットワークの中核として、成長のためにリソースを解放し、ITの将来に向けた 革新的なテクノロジーの創出を支援しています。

アジア太平洋
+65 6490 4200

オーストラリア
1 800 733 428

ブルネイ / カンボジア
800 862 6691

インド
+91 22 3987 8888

インドネシア
001 803 440224

日本
03 5798 8510

韓国
080 708 0880

マレーシア
1 800 812 678

ニュージーランド
0800 450 503

フィリピン
800 1441 0229

シンガポール
800 448 1430

タイ
001 800 441 6039

ベトナム
800 862 6691

中国
800 810 2100

香港
852 3002 1362

台湾
0800 666 052

facebook.com/redhatjapan
@redhatjapan
linkedin.com/company/red-hat

jp.redhat.com
#INC0410962_v1_201606_KVM