CoreCenter の採用技術

Adopted Technology of CoreCenter

原　　加奈恵

要　約　次世代流通ソリューション CoreCenter® は，機能選択型の導入型ソリューションとして，顧客の変化に対応するパッケージを目指し開発した．高品質・短納期・低価格といったマーケット要求に応えるべく，コンポーネント化と SOA を志向し，様々な変化に柔軟に対応するためのパッケージの構造を定義するとともに，導入方法論も再定義している．本稿では，CoreCenter が採用したパッケージ構造の特徴と，導入方法論について解説する．

Abstract　CoreCenter® is next-generation solution package for the distribution industry. This package is “Implementation-based solution”, it can be assembled by selecting the services. It was developed with the aim that package can respond to various changes of customers. To meet the requirements of the market-place such as high-quality, short delivery, and low price, “component oriented Architecture” and “SOA” are adopted, and package structure is defined to have a structure that can respond flexibly to various changes. At the same time, We redefined the implementation methodology. This report describes the implementation methodology and the features of package structure adopted for CoreCenter.

1. は　じ　め　に

CoreCenter® シリーズは，日本ユニシス株式会社（以下，日本ユニシス）が開発した次世代統一 AP 基盤 CoreCenter BASE と，その上で動作する流通各業態向けのソリューションパッケージで構成される．

CoreCenter は，高品質，短納期，低価格といったマーケット要求に応えるべく，これまでの各顧客個別にカスタムメイドでアドオンを前提としたパッケージ提供の形態ではなく，顧客ごとの違いにも対応できる機能選択型の導入型ソリューションとして開発された．顧客の業務要求は多岐にわたるが，QCD を維持した上でパッケージを導入し運用を開始するためには，顧客は，パッケージ自体の理解を深め，なるべく追加開発をしないアプローチが必要である．また稼働後においても，顧客の業務変化や法改正に応じた迅速な対応が求められ，その際高い品質で柔軟な対応を継続できるよう，パッケージ機能で変化に対応する仕組みが必要である．

本稿では，導入型ソリューションとして開発した CoreCenter シリーズの中から，小売業向けパッケージである CoreCenter for Retail で採用したアーキテクチャと導入方法論を解説する．2 章では，パッケージ機能に対し，変化に対応するために採用したコンポーネント化と SOA 適用の概念について，3 章で，短期間で導入するためのアプローチとして定義した導入方法論を解説する．

2. アーキテクチャ

本章では，CoreCenter for Retail を効率的に適用/導入するために採用しているアーキテクチャについて解説する．

2.1 CoreCenter アプリケーションの論理構造

CoreCenter for Retail のアプリケーション構造を図 1 に示す．この構造は CoreCenter BASE で規定しており，各層の役割を明確に定めている．役割ごとに論理階層（レイヤ）を分け，レイヤ間の依存関係を小さくすることで，あるレイヤの実装を変更しても他のレイヤに影響を与えない構造である．各層の役割を以下で説明する．

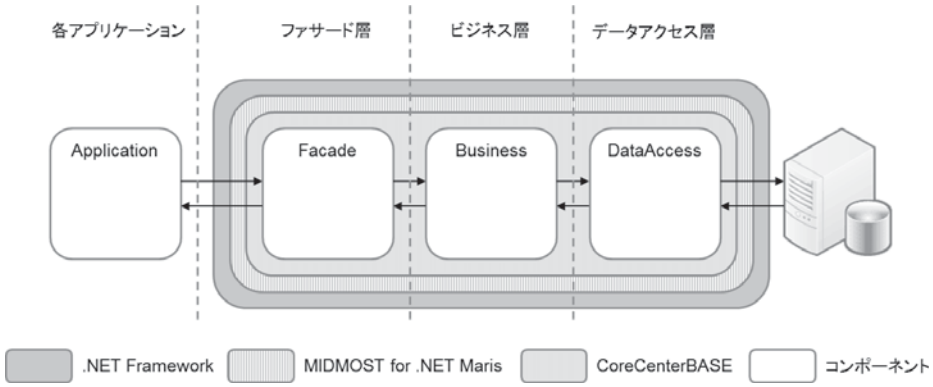

図 1　CoreCenter アプリケーションの論理構造

1）　ファサード層

トランザクション管理とビジネスの業務処理の呼び出しを担当する．アプリケーションからの呼び出しはすべて Façade が受け取り，一つ以上の Business を呼び出し，結果を各アプリケーションに返す．Façade の 1 処理がトランザクションの単位となる．Façade にはビジネスロジックを記述せず，共通的な処理の実装にとどめることで，ビジネスロジックの冗長化を排除している．

また，Business の呼び出しには，CoreCenter BASE が提供する DI（Dependency Injection）を利用している．これにより，既存プログラムを変更せずに，外部ファイルの変更で処理内容を切り替えることが可能である．

2）　ビジネス層

業務処理を担当する．Business は，DataAccess とデータのやりとりをしながら業務処理を実行し，結果を Façade に返す．Business 同士の呼び出しも原則 DI を経由する構造である．

各サブシステムは，その境界に公開用クラスを提供し，他サブシステムの Business を呼び出す必要がある場合は，公開用クラスのみ利用可能としている．公開用クラスは，サブシステムの独立性を保つために定義しており，CoreCenter for Retail 内での循環参照を防止するとともに，サブシステム単体導入時に利用可能なインタフェースを提供する．

3）　データアクセス層

データベースに対する操作を担当する．DataAccess は，データベースからのデータ取得，データ更新を実行し，結果を Business へ返却する．ここで SQL を隠蔽している．

各サブシステムの DataAccess は，自サブシステム内のテーブル操作のみ許可する．他サブシステムの情報が必要な場合は，前述の公開用クラスを介すルールとすることで疎結合化を図っている．

CoreCenter for Retail が提供するアプリケーション形態は画面・バッチ・Web サービスであるが，どの場合でも，ファサード層以下の構造が同じになるように作成することを前提としている．アプリケーション形態に関わらず業務プロセスをモデル化し，実装と対応付けることで，再利用性を向上させるためにこの構造を採用している．同時に，保守性も向上させることが可能となる．

2. 2　CoreCenter アプリケーション構造の特徴

CoreCenter for Retail を顧客へ適用し業務運用を遂行する際，顧客ごとの業務ルール等の違いに対応する方法として「カスタマイズ」と「アドオン」がある．CoreCenter for Retail では，顧客が利用する機能やパラメータにより処理を選択することを「カスタマイズ」，顧客独自仕様を機能に盛り込むことを「アドオン」と呼ぶ．

前節で述べた論理階層に沿ったアプリケーション構造で，カスタマイズを実現し，アドオンを効率的に実施するための構造面の特徴を本節で説明する．

2. 2. 1　コンポーネント化

顧客が選択できる機能やパラメータで選択可能な処理が多いほどアドオンを減らすことは容易になるが，同時に選択しやすい粒度の検討が必要である．粒度が細かすぎるとパッケージ全体が複雑化し，再利用できる部品を見逃し，再利用が徹底できない恐れがある．また，DI の仕組みを有効利用するためには，差し替えしやすい粒度のコンポーネントとする必要がある．つまりビジネスコンポーネントの粒度は，カスタマイズで選択しやすい単位であること，さらにアドオンも意識して差し替えしやすく，かつ再利用性が高い粒度である必要があり，導入型ソリューションと位置づけるために重要な概念となる．

その粒度は，これまで日本ユニシスが導入してきた小売業基幹システムと，そのアドオン事例を分析し決定している．アドオン内容から可変ポイントを分析しコンポーネント候補とした後，業務の変化が発生する単位に統合・分離する．パターン化できるアドオン内容はパラメータ制御によりカスタマイズを可能とする．各顧客固有のロジックが入りやすい業務は，コンポーネントを分割し，差し替えしやすい単位にまとめ，アドオンによる作りこみ部分を極小化できるようにする．逆に業務変化に依存しないコアな部分は統合していき，コンポーネントの利用者が意識するインタフェースを減らすよう考慮した．結果 CoreCenter for Retail は，全体でコンポーネント数約 5000，パラメータ数約 1600 で構成している．

図 2 にコンポーネント構成例を示す．分析結果から導き出した切り替え単位を「クラス」とし，各コンポーネントを割り当てることで，DI による差し替えが可能な単位を定義している．

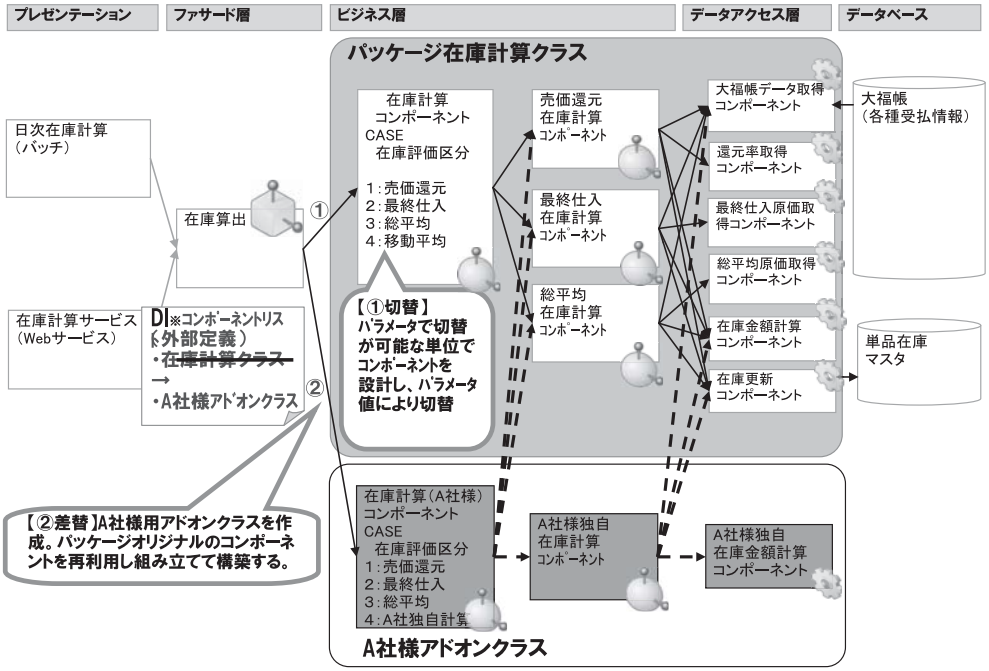

図 2　コンポーネント構成例

コンポーネント粒度や部品構成の見直しにより，導入時の効率化のみでなく，導入後の業務変化にも柔軟に対応できる仕組みとなり，顧客へ導入した仕組みを継続的に改善できる基盤が整ったと考える．

2. 2. 2　SOA の適用

SOA（Service-Oriented Architecture）はビジネスレベルの「サービス」を組み合わせてアプリケーションの連携や統合を行うシステム構築の考え方である．前項のコンポーネントは機能間での再利用性向上を目的としているのに対し，サービスは，より大きな粒度で再利用し，複数の外部システムと連携して業務プロセスを実現する．CoreCenter for Retail は，小売業のマーチャンダイジングサイクルを全 11 のサブシステムで構成しており，各サブシステムを「サービス」と捉え SOA を志向している．

各サービスは単独で稼働し，サービス内は独立した機能のまとまりで業務プロセスを構成する．サブシステム外の業務プロセスは，それぞれサービスインタフェースを介すことで実現する．この構造により，顧客要求に沿ったサブシステム単位の部分導入や，優先度の高い業務から段階導入するようなスモールスタートが可能となるメリットがある．また，今後のマルチチャネルからのアクセスへの対応にも有効である．

各サービスに必要なサービスインタフェースは，図 3 に示す SIM（System Interaction model：システム相互作用モデル）を用いて境界を示し，定義している．

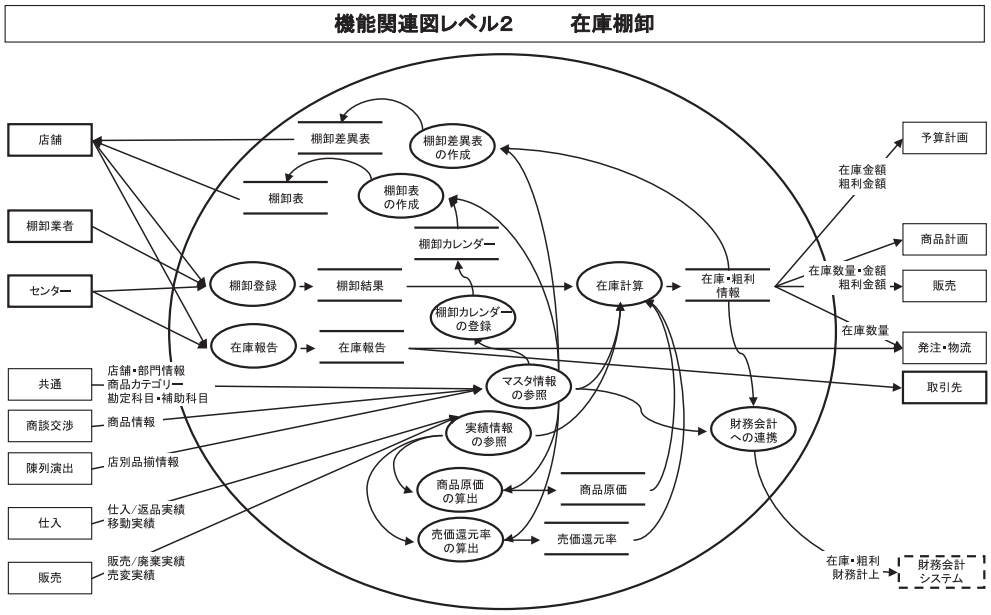

図 3　SIM 例

各サービスインタフェースは図 4 に示すとおり，様々な外部システムとの連携を想定し，それぞれファイルインタフェース，Web サービス，API，View の 4 種類のインタフェースを用意することを基本としている．

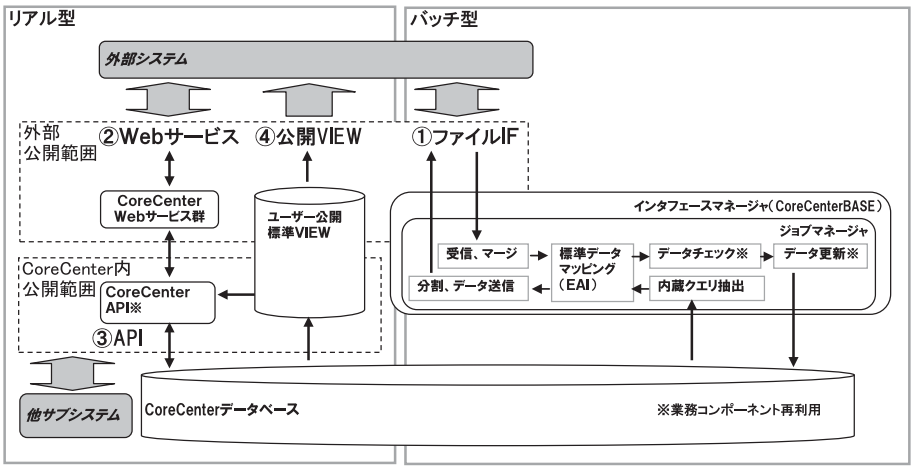

図 4　サービスインタフェース概観

外部システムはファイルインタフェースによるバッチ型の連携方式が多く，ほとんどの場合，CoreCenter for Retail 導入時に現行の連携方式を継続することが要件であるため，ファイルインタフェースを選択することとなる（図 4 の①）．この場合は，CoreCenter BASE の EAI 機能を用いることでパッケージが定義している形式へ変換することができるため，業務ロジックを変更せずに，顧客から指定された連携方式に対応することが可能である．

外部システム側が変更対応することが可能な場合や新たに追加するインタフェースは，リアルタイム連携が可能なWebサービス型を選択する（図4の②）．Webサービスは，CoreCenter for Retail内のサブシステム間接続で用いる公開用クラス（API：図4の③）を用いて，外部システムへ公開可能な項目でインタフェースを定義し実現している．

CoreCenter for Retailに格納された情報の参照には，Webサービスの他に公開View（図4の④）を利用することが可能であり，外部システム側の処理で利用することや，帳票や検索機能を顧客側で自由に追加することができる．

3. インプリメンテーション

本章では，CoreCenter for Retailを効率的に適用/導入するために必要となる導入方法論について解説する．前章のコンポーネント化とSOAを志向した構造により実現できる導入方法であり，Profiling（業務分析），パラメータ設計/設定および実機検証の各作業を，従来の顧客業務ヒアリング中心から，パッケージが持つ機能で構成する業務を標準とし，その業務を顧客が理解し，運用を合わせていくプロセスに再定義した．

3.1 導入方法論「CoreMethod」の策定

CoreCenter for Retailの導入方法論は「CoreMethod」と呼び，高品質・短納期・低価格を目指す導入プロセスとして以下の3点を考慮し策定した．

1） 顧客のCoreCenter for Retailの習熟度を向上させること
2） アドオン開発を削減すること
3） 要件確認漏れによる手戻りを削減すること

3.2 従来のパッケージ導入との違い

CoreCenter for Retail以前のパッケージ導入プロセスは，日本ユニシスが顧客業務を理解することから開始していた．AsIsを聞ききってからToBeを検討することとなり，これにより現行業務の聞き過ぎがアドオン増加の原因となり，また，聞き漏らしが起きれば手戻りの原因ともなった．

そこでCoreMethodでは，CoreCenter for Retailが用意する標準業務プロセスをToBeとして，顧客の理解をより深めるプロセスを定義した．顧客の理解を醸成しつつ要件を拾い上げ，カスタマイズ（パラメータ設定）を決定するアプローチとなる．

また，顧客による実機検証は，従来はシステムテスト工程での実施であったが，CoreMethodでは従来のウォーターフォール型開発で言う上流工程で実施する．仮決定したパラメータを設定し，顧客が業務フローに基づき実機を用いて検証することにより，顧客のCoreCenter for Retailの習熟度を早期に向上させ，後工程での要件/仕様齟齬を削減することを目的としている．

3.3 導入プロセスの定義

CoreMethodの導入プロセスは，図5のとおり3フェーズで構成している．CoreMethodでは各フェーズ内でタスク順序，タスク内容などを詳細に定義している．本節で各フェーズの概要を解説する．

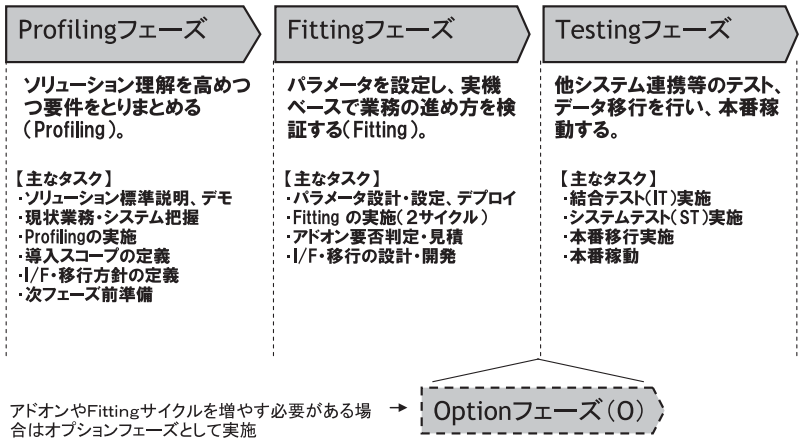

図 5　CoreMethod の構成

3. 3. 1　Profiling フェーズ

顧客のパッケージへの理解を高めつつ要件をとりまとめるフェーズであり，ウォーターフォール型開発プロセスでの要件定義（Fit & Gap）に相当する．

当フェーズは，パッケージが持つ業務範囲を想定業務フローに沿って解説した「標準業務説明書」（図 6）を中心に進める．標準業務説明書はサブシステム別にあり，その構成は，顧客の理解度に応じ概要から詳細まで少しずつ学習できるよう，サブシステムの概要と基本事項を解説し，想定業務フローへとつながっている．業務フロー内の業務はそれぞれ作業手順が確認でき，必要に応じてさらに詳細なビジネスルール定義書を参照することができる．Profiling 開始時にこの資料を顧客に提供し，一定回数の打ち合わせの中で，標準パラメータ設定によるデモ，ヘルプ機能を用いて CoreCenter for Retail の業務/機能を解説しながら顧客の理解を促す．この中で要件をとりまとめ，標準機能の要/不要，標準業務フローの見直しを行い，大まかなパラメータ設定を決定する．パラメータ設定では対応できない業務要件はアドオン候補として洗い出し，QCD への影響を顧客と共有するとともに，CoreCenter for Retail による代替運用を検討する．

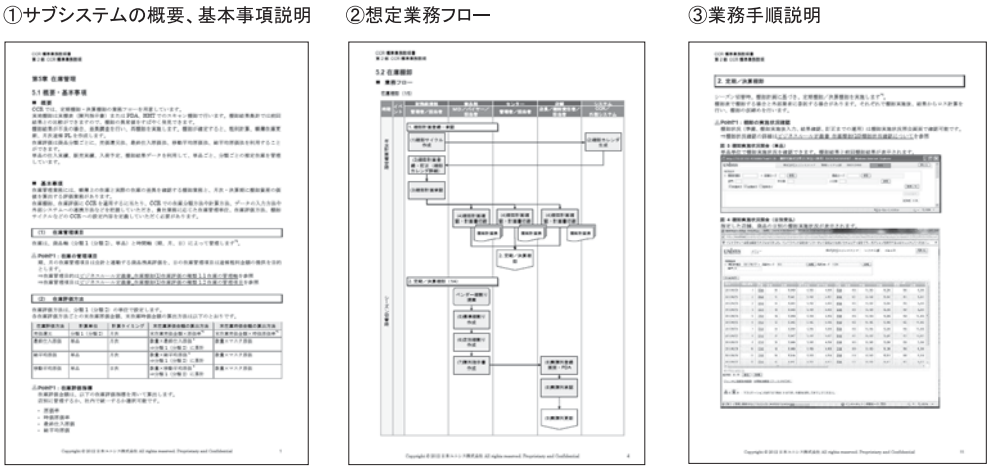

図 6　標準業務説明書イメージ

3. 3. 2　Fitting フェーズ

パッケージへの理解を深めつつパラメータ設定を決定するフェーズであり，ウォーターフォール型開発プロセスでの論理設計に相当する．

当フェーズでは Profiling フェーズで仮決定したパラメータを設定し，顧客向けに実行形式化する．ここで実施するパラメータ設定で，2. 2. 1 項で述べたビジネスコンポーネントのカスタマイズが可能である．また，画面や帳票の見た目，CoreCenter for Retail の UI コントロールが対応している可変項目（単項目チェックなど）にも対応する．このパラメータを設定した機能を用いて，顧客は，業務シナリオに基づいた実機検証を行い，Profiling フェーズで検討した CoreCenter for Retail による代替運用もこのときに確認する．実機検証の中で抽出された要望は，パラメータ設定で対応可能な範囲で再設定し，再度実機検証を行う．実機検証はこの 2 サイクルで完了とし，アドオン範囲を確定することで当フェーズを終了する．

当フェーズで，顧客が運用開始時に近い状態で機能を使用することにより，顧客と日本ユニシスとの間の認識のずれを抑えることができ，要件確認漏れによる手戻りの削減につながる．このプロセスはアジャイル開発手法のイテレーションと類似しているが，実機検証時の確認ポイントを事前に顧客と共有し，完了基準も明確にすることで，2 サイクルでの完了を可能としている．

3. 3. 3　Option フェーズ

アドオン候補の論理設計は，Fitting フェーズで行い，最終的にアドオンが必要と判定した機能やビジネスコンポーネントに対し，この Option フェーズを実施する．このフェーズでアドオンに対し，ウォーターフォール型プロセスでの物理設計/開発/単体テストに相当する作業を行う．なお，Fitting で追加の実機検証が必要と判定した場合も，Option フェーズで対応する．

アドオン開発は，パッケージが持つコンポーネント構造を考慮し，標準コンポーネントを最大限再利用するよう設計する．また，ビジネスコンポーネントのアドオンは DI による差し替えを利用するため，局所化した対応となる．コンポーネント化は Option フェーズでの開発作業の効率化にも有効である．

3. 3. 4　Testing フェーズ

外部システム連携の検証，データ移行を行い本番稼働するフェーズであり，ウォーターフォール型開発プロセスでの結合テスト/システムテストに相当する．アドオンを含めた顧客利用機能全体で，移行データによる稼働確認とインタフェースを含めての最終テストを実施する．

このとき，アドオンが関連する範囲を標準テストケースから抽出し結合テストを実施するため，Testing フェーズ内で従来の結合テストが占める割合を抑え，システムテストを充実させることが可能である．

3. 4　各工程での注意点

CoreMethod は，CoreCenter for Retail を ToBe として顧客の業務をパッケージ業務に合わせていくアプローチである．顧客と取り決めた QCD を実現するため，顧客もその認識を強く持つ必要があるとともに，顧客が実現したい業務を日本ユニシスが正確に捉え，パッケージの機能で実現する代替案の提案が不可欠である．

さらに，この導入プロセスは顧客のパッケージへの理解度を早期に深めることが重要であり，それができなければ顧客に納得感のある導入が困難となる．そのためパッケージの理解を深めるためのプロセスと，各フェーズでの顧客の習熟度の計り方は工夫が必要であり，互いに共有していく必要がある．CoreMethod では，Profiling から Fitting までで理解を深め，その後稼働までに「人に教える」レベルに到達するような習熟度曲線を定義している（図 7）．

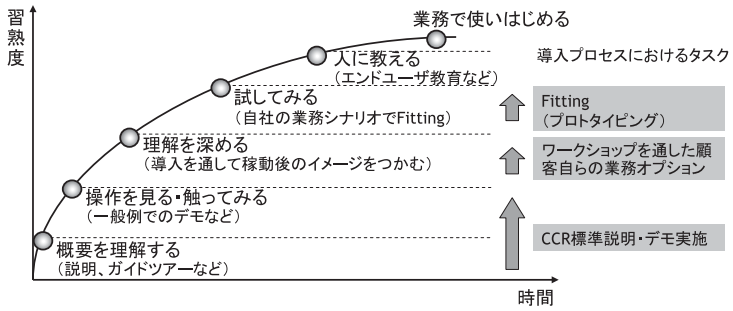

図 7　各工程での習熟度

習熟度は，表 1 を用いて各プロセスで顧客自身が点数化し，各工程終了時に日本ユニシスとともに判定する．評価点数は，3 段階（1：説明に従い操作した，2：担当範囲の実機操作を理解した，3：業務の流れを確認した）で自己評価する．合格ラインはサブシステムごとに 2 点の合計得点とし，2 点に満たない項目は個別対策などを実施し習熟度の向上を図っている．ただし，各工程で到達しておくべき点数や，点数基準の妥当性は模索中である．パッケージ適用を重ねる中でブラッシュアップし，より効率的な導入を目指していく．

表 1　理解度判定表の例

確認グループ	確認概要	確認No	確認内容	確認ポイント	確認日	確認者	理解度（自己診断）
商品登録（雑貨）	商品マスタ新規登録シナリオ	1		商品マスタ新規登録の流れを理解する。			
	新規登録	1-1	■商品マスタ情報の新規登録 商品マスタメンテナンス画面にて商品登録する。 ・グロサリの仕入販売商品	・商品マスタの項目と設定方法を確認する。			
	承認	1-2	■上記で新規登録した商品を承認する。 ・承認依頼商品リスト画面	・承認の流れと、商品のステータスの意味を確認する。			
	店別原価売価の設定	1-3	■商品の店別での、売価・仕入条件設定の実施 ・登録、確認済み商品の店別原価売価を変更。 ①店別原価売価検索画面から、 有効開始日を指定し、対象商品を検索 ②店舗グループを利用しての 売価・仕入条件の一括設定 個店指定での売価・仕入条件の設定も可能。	・店舗グループに紐付く店舗に対して、予め売価・仕入パターンが設定されていること。 ・店別の売価・仕入パターン設定方法を確認する。			
	複数スキャンの設定	1-4	■スキャニングマスタメンテ画面で 1商品に複数JANの設定を実施 ①登録・確認済みの商品の スキャニングマスタを検索する。 ②新規でJANコードを追加する。	・複数JANの設定方法と利用時の動きを確認する。			

4. お わ り に

CoreCenter for Retail のリリース以降，数社へ適用している．本稿で解説したアプリケーション構造と導入方法論によって，以下の効果があった．従来型と比較して効率的に適用が進んでいると考える．

・Profiling 終了時点で，アドオン削減の方向に進めやすくなった．
・Fitting 時の実機検証に対し，顧客の協力が得やすくなった．
・Fitting 以降業務レベルでの齟齬の発生が減り，齟齬が発生した場合も追加開発の規模が削減された．
・Option フェーズ（パッケージ機能に対するアドオン量）が削減された．

今後はその「効率」をより定量的に捉え，各顧客へ適用したアドオンからパラメータ化を推進するとともに，標準ビジネスコンポーネントを拡充し，パッケージの成熟度の更なる向上を目指している．

執筆者紹介　原　　加奈恵（Kanae Hara）
1999 年日本ユニシス(株)入社．以来，小売業を中心としたシステム構築，ソリューション適用に従事．2009 年より CoreCenter 開発を担当し，現在は CoreCenter for Retail の顧客適用に従事している．