開発方法 LUCINA for .NET ——開発プロセスとアーキテクチャの策定

LUCINA for .NET ——Development Process and Decision of Architecture

白 木 和 彦, 幸 野 俊 介

要 約 近年のシステム開発は，次々に登場する新しい技術やパッケージ製品などの選択肢が多い反面，それらを組み合わせて，システムの要求を満たすためのアーキテクチャを決めることが困難な状況にある．また，開発の具体的な作業とその順序や成果物を定義する開発プロセスについても，短期開発，開発期間中の変更などを前提とする必要がある．日本ユニシスの開発方法である LUCINA for .NET は，企業システム全体の最適化を目指した，.NET による Web アプリケーション開発のアーキテクチャと開発プロセスを定義しており，このような状況の中でプロジェクトを成功に導くことを目的として作成された．

本稿では，LUCINA for .NET の開発プロセスの概要を紹介した後，実際の開発で行われる作業内容やアーキテクチャ策定の内容について述べる．

Abstract In the system development of recent years, new technologies and package software, which appear one after another, give us more choices in system development. This, however, makes it difficult for us to consider combination of them and to determine the architecture which satisfies system requirements. Moreover, in the development process where we define specific tasks, the sequence among them, and deliverables, the short-term development and whatever changes to occur in the process of development should be the major premises. "LUCINA for .NET" is the development methodology of Nihon Unisys, which defines architecture and development processes of the Web application development with ".NET" that aims to optimize the entire corporate system. This methodology was created in order to bring a project to success in such a situation today.

This paper introduces the overview of the development process of "LUCINA for .NET", and then discusses the actual tasks or how to develop the architecture.

1. はじめに

システム開発を取り巻く環境はさまざまであるが，一般的に，開発を進める上で重要な要素は，以下の 2 点を早期に明確化して開発者に周知することであり，新しい環境で開発する場合は特に重要である．

1) アーキテクチャ[*1]（設計方針）
   - 品質に関する要求を満たすために安定した技術を見極めて設計の方針を立てること
   - 一貫した考え方のもとで実現されるシステムの設計を確立すること
2) 開発プロセス
   - 開発作業の進め方と成果物を定義すること
   - 開発の計画や進捗状況が管理できるようにすること

日本ユニシスの開発方法である LUCINA for .NET は，企業システムにおける .NET での

Web アプリケーション開発を前提として，アーキテクチャと開発プロセスを定義している．ただし実際の開発ではシステムの品質に関する個別の要求があり，それを満たすためにプロトタイプによる評価，検証を実施し，プロジェクト固有のアーキテクチャを確定する作業（以下，アーキテクチャ詳細化）が必要である．開発プロセスについても実際の開発のスケジュールや体制などを踏まえ成果物名称や記述内容，書式を定義する必要がある．

本稿では，LUCINA for .NET を適用した開発で実際に行う作業について述べる．2 章では，開発プロセスの概要と特長について紹介する．3 章と 4 章では，LUCINA for .NET で定義している二つの開発フェーズで，実際にどのように開発していくのかについて述べる．

2. LUCINA for .NET 開発プロセス

本章では，LUCINA for .NET が定義する開発プロセスについての概要と特長について紹介する．

2.1 LUCINA for .NET 開発プロセス概要

LUCINA for .NET の開発プロセスは，「コンポーネント・システム[*2]」と呼ばれるサブシステムの外部仕様（インタフェース）を導出するまでの「業務分析フェーズ」と，その内部仕様を設計して実装する「システム構築フェーズ」の二つで構成される（図 1）．短期開発では要求を早期に実装できるよう開発フェーズを少なくすべきであるが，インタフェース設計と実装は分ける必要があるため，インタフェース設計を確定する「業務分析フェーズ」と内部を実装する「システム構築フェーズ」とに分けている．実際の開発では開発現場のフェーズ名称に置き換えることも多い．

また短期開発に限らず，「業務分析フェーズ」を要件定義・論理設計（または基本設計），「システム構築フェーズ」を物理設計（または詳細設計）・実装/単体テスト・結合テスト・統合テスト・システムテストの作業に対応させ，それぞれ数か月単位で開発するような大規模開発でも，LUCINA for .NET の開発プロセスを適用して成功した実績がある．

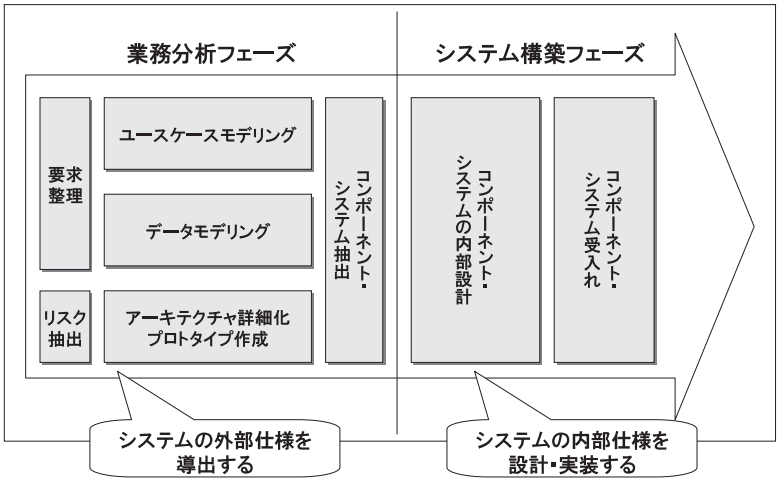

図 1　LUCINA for .NET 開発プロセス概要図

2.2 LUCINA for .NET 開発プロセスの特長

1) ユースケース駆動

ユースケース駆動とは，利用者（アクタ）からみたシステムの使い方であるユースケースを起点に開発作業を進めるアプローチである．ラショナル統一プロセス （以下，RUP）でも採用されている考え方であり，ユーザの要求を念頭において開発作業を進めることができ,構築したシステムとユーザ要求との間のズレを回避することができるため, LUCINA for .NET でも採用している．

2) アーキテクチャ中心

アーキテクチャ中心とは，開発時の技術的なリスクを回避するために，システムに求められる要求やリスクを入力とし，プロトタイプによる技術検証を実施して，開発の早期段階でシステムの骨組みであるアーキテクチャを確立することである．アーキテクチャ中心も RUP で採用されている考え方である．更に LUCINA for .NET では.NET プロファイルを用いて企業の Web アプリケーションを構築する際の具体的なアーキテクチャを定義している．このため LUCINA for .NET を適用した開発では，アーキテクチャを最初から定義するのではなく，LUCINA for .NET が定義したアーキテクチャをベースとしてプロジェクトの個別の要求に応じて詳細化することができるため,開発期間の短縮が可能となる．

3) データ中心アプローチ

データ中心アプローチ（DOA : Data Oriented Approach）はシステムで利用するデータに着目してシステム分析/設計を行うアプローチである．変更に強い安定したシステムを構築することができ，RDBMS との整合性が高く，Visual Studio .NET の高生産性を最大限に発揮することができるため，LUCINA for .NET では DOA を採用している．

3. 業務分析フェーズ

業務分析フェーズで実施する作業について説明する．

3.1 要求整理

要求整理では，顧客の要求の中で不足部分や曖昧な部分に関して，顧客や関係者と折衝して整理する．整理した結果を要求記述としてまとめる．要求整理には，業務的な側面の機能要求とシステム的な側面の非機能要求がある．アーキテクチャを確立するために重要なのは非機能要求であり，例えば表 1 のような項目が考えられる．

要求整理，要求記述は LUCINA for .NET の用語であり，一般的にはそれぞれ，要件定義，要件定義書と呼ばれる．本稿では LUCINA for .NET の用語で記述する．

3.2 ユースケースモデリング

システムが提供する機能をユースケースとして外側から捉え，ユースケースを詳細化したシナリオ，画面遷移と分析していき画面や帳票の入出力を洗い出していく．

3.2.1 ユースケース抽出

システムに必要な機能を，ユースケースとして洗い出す．アクタとの関係も明確にする．開発者の視点ではなく利用者の視点で抽出することが重要である．ユースケース抽出によりシステム化対象範囲を明確にすることができる．

表 1　非機能要求項目例

分類	大項目	小項目	概　　要
品質	性能	キャパシティ	ユーザ数，データ量，スループットなどシステムが想定する最大容量に関する要求
		パフォーマンス	平均応答時間に関する要求
		スケーラビリティ	スケーラビリティを確保するための要求
	信頼性	信頼性	システムの障害発生を最小限に抑えるための要求
		可用性	システムを長時間使用するための要求
		保守性	保守の容易性に関する要求
		セキュリティ	安全性や機密性に関する要求
	運用管理	サービス監視	サービスを監視して目標値と比較するための要求
		システム管理	システムの開始・終了，バックアップ・リカバリなどのシステム管理に関する要求
		ソフトウェア配布	ソフトウェアの配布方法や手段についての要求
	ユーザビリティ		画面遷移やユーザナビゲーションなど，ユーザの操作性に関する要求
制約	ソフトウェア		システム構築の前提となるソフトウェアについての要求
	ハードウェア		システム構築の前提となるハードウェアについての要求
	既存システム		システム構築の前提となる既存システムとの連携に関する要求
	プラットフォーム依存性		システムのプラットフォームに対する依存性に関する要求
	環境や実装の制約		開発規約や実装時のポリシー，法的な規約に関する要求

3.2.2　シナリオ抽出

ユースケースに対して，アクタとシステムの間のやり取りを詳細に分析してシナリオを洗い出す．その後で，画面遷移図，画面の入出力項目と画面動作，帳票の出力項目を洗い出す．画面の入出力項目や帳票の出力項目に対して，入力形式，入力チェックの仕様，デフォルト値，表示形式などを明確にする．

ここで重要なのはユースケース，シナリオの後で具体的な画面，帳票を設計することである．システム利用者の目に触れる画面や帳票のイメージを最優先に設計するあまり，ユースケースやシナリオを後回しにすると，不要な画面が存在する（または必要な画面が存在しない）などの手戻りが発生する．「誰がどう使うか」を確定してから「どんな画面」を確定することで，開発の手戻りを減らすことができる．また，ユースケースやシナリオは承認（アクセス制御）の設計の入力情報となることからも早期に確定させる必要がある．

3.2.3　画面/帳票設計

プレゼンテーション部分の設計を実施する．レイアウトなどの詳細にこだわらずに，入出力

する情報の種類，画面遷移などを定義する．

3.3 データモデリング

一般的なデータモデリングの方法と同様であるが，LUCINA for .NET ではユースケースやシナリオに着目しているため以下のように変更している．

3.3.1 概念データモデル定義

システムで扱われるデータを概念レベルで明確にする．テーブルの意味，認識を主に表現し，テーブル間の関係を明確化することによってデータ構造の分析を行う．

ユースケースに含まれるアクタ・物・データ・帳票・トランザクション等を抽出し，これらの情報をアクタ・トランザクション・マスタに分類して関連付けを行う．ユースケースは自然言語で記述されるので，その曖昧さからデータ項目の関連に矛盾を感じる場合があるが，そのような場合にはユースケースを見直して“何を永続するか”に観点を置いて再考する必要がある．

3.3.2 論理データモデル定義

概念データモデルで抽出したデータ項目に，画面項目定義やシナリオから属性を割り当て，テーブルを作成する．すべての項目ではなく，全体の 70%〜80% の属性を定義すればよい．

さらにそれぞれのテーブルの属性を分析し，第 3 正規形まで正規化を行う．テーブル間の関係を入れた ER 図を作成する．

3.4 コンポーネント・システム抽出

コンポーネント・システムは，業務データの固まりを単位に抽出する．コンポーネント・システムの粒度は企業システム全体のレベルで捉えたときのコンポーネント（部品）であり，サブシステムの単位とするのが現実的である．コンポーネント・システムは，トランザクションの同期を保証する業務データの集まりを内部に隠蔽する．

3.4.1 コンポーネント・システム仕様

コンポーネント・システムが確定したら，次にコンポーネント・システムが外部に対して公開する機能（サービス）を洗い出す（図 2）．ユースケースやシナリオを分析した結果出てくる画面や帳票が，業務データに対してどのような情報を渡し，どのような情報を期待しているのかを一覧表にまとめていく．期待している情報が業務データそのものだけでなく何らかの加工（計算や判断）が必要であれば，それがコンポーネント・システム内部で実装されるビジネス・ロジックとなる．このような作業をすべてのユースケースに対して実施すると，コンポーネント・システムが外部に公開する機能（サービス）をすべて洗い出せる．機能（サービス）の妥当性と業務データの過不足を相互にチェックすることでビジネス・ロジックの仕様が明確になっていく．

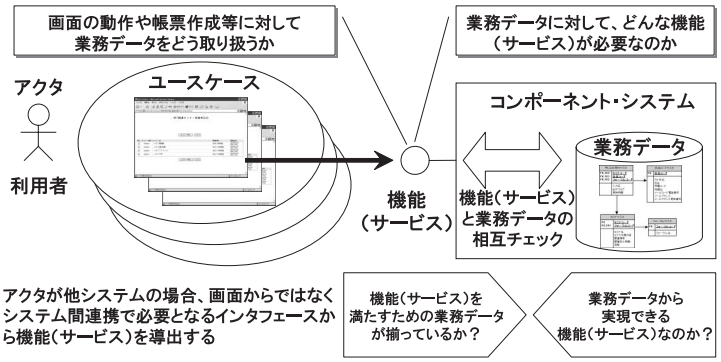

図 2　コンポーネント・システムの機能（サービス）の洗い出し

3.5　リスク抽出

要求記述からリスクとなり得る要求を抽出して一覧を作成する．統一した考え方でシステムを構築できるよう，通常は技術的なリスクの解消を優先して行う．

リスクは早期に解消する必要があるため，優先度を定め，その高い順に調査・検証を実施する．各リスクの検証結果，検証実施担当者などを管理し，確実にリスクを解消していく．

3.6　プロトタイプ作成

アーキテクチャの実現可能性を調査するため，プロトタイプによる技術的検証を実施する．プロトタイプの目的や前提，検証箇所などをプロトタイプ実施計画書にまとめ，プロトタイプの検証結果はプロトタイプ結果報告書にまとめる．

プロトタイプ作成は 1 回とは限らず，リスクの難易度やプロトタイプの目的に合わせ，数回実施することもある．必要なときに適宜実施し，その検証結果はアーキテクチャ設計に随時フィードバックする．

また，完全な検証は本番環境でしか実施できないケースもあり，どのような前提でどこまで確認したのかを後で参照する必要があるため，検証環境と本番環境との相違点がある場合は明確にしておく．

3.7　アーキテクチャ詳細化

システムを設計していく際の基本方針を定めるアーキテクチャは，要求整理の結果と LUCINA for .NET で事前に定義されたアーキテクチャをベースに，プロトタイプによるリスク検証を繰り返しながらプロジェクト固有のアーキテクチャを詳細化して確定する．

アーキテクチャを詳細化するための一般的な項目の例を表 2 に示す．

3.7.1　開発テンプレート作成

システム構築フェーズでの生産性向上および品質の向上と均一化を最も効果的にするため，開発テンプレートを作成して，開発時のルールを規定し，開発者に対して周知，徹底する．開発テンプレートには設計書やサンプルコード，クラスライブラリやその解説書等がある．プロジェクト内の開発者は，個人の開発経験に依存してしまいがちな設計や実装のノウハウを共有することができる．

表 2　アーキテクチャ詳細化の項目例

大項目	小項目	内　　容
論理構成	情報保持方法	画面間の情報保持方法
	入力チェック方法	ユーザの入力内容のチェック箇所と方法
	認証方式	認証方式の決定
	承認方式	ユーザ権限に関する情報保持と仕組み
	トランザクション方式	トランザクション制御の方針決定
	同時実行制御	DB 更新処理に関する排他制御の方針決定
	バッチ連携方式	バッチ処理の連携方針決定
	DB アクセス方式	DB アクセス方法の決定
	セキュリティ	セキュリティ実現方法（認証・承認，コーディング上の注意点，各種設定）の決定
	メッセージ管理	メッセージの一元管理やメッセージ表示に関する方針決定
	トレース	トレースの仕組みと出力方針決定
	帳票実現方式	帳票出力の方針と実装方針決定
	例外処理方式	例外発生時の処理に関する統一した考え方や方針決定
物理構成	サーバ構成	サーバ構成図を使ったサーバ配置検証
	ネットワーク構成	ネットワーク構成図を使ったネットワーク・トラフィック検証
	負荷分散・スケールアウト方式	ロードバランスやクラスタリングの方針決定
	インフラレベルセキュリティ	ファイアウォールへの設定内容決定

4.　システム構築フェーズ

システム構築フェーズでは，コンポーネント・システムの内部を設計して実装する．

4.1　コンポーネント・システムの内部設計

LUCINA for .NET では，コンポーネント・システム内部の論理構造として図 3 で示すようにビジネス・ロジック層/データアクセス層/データ層の 3 層構造をとっている．また，四つのタイプのオブジェクト（《control》,《business》,《target proxy》,《data access object》）に分割している．さらに，データの受け渡しの入れ物となるオブジェクトとして《data transfer object》を，コンポーネント・システムを操作するユーザ・インタフェースである「フロントエンド」とコンポーネント・システムとで共通に使用する．

3.4 節で洗い出した機能（サービス）のビジネス・ロジックをビジネス・ロジック層で実装する．具体的には《control》で実装するが，複数の《control》で実装する処理は《business》で実装して《control》から呼び出す．《target　proxy》は別のコンポーネント・システムを呼び出すときに使用する．《data access object》は，RDBMS のデータを《data transfer object》に入れたり，《data transfer object》のデータを RDBMS に反映したりする．

4.1.1　内部インタフェース設計

内部インタフェース設計では,《control》,《business》,《target proxy》,《data access object》の四つのそれぞれで実装する処理のインタフェースを設計する（図 3）.

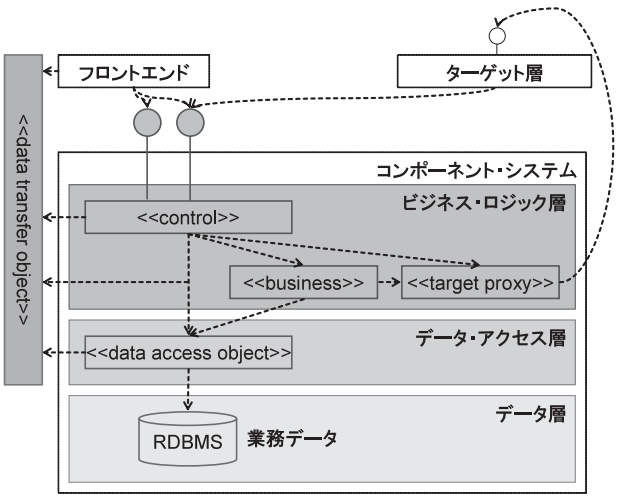

図 3　コンポーネント・システム内部の論理構造

4.1.2　クラス仕様設計

内部インタフェース設計に対するクラス内部の仕様を定義する．

クラス図からクラス仕様を作成し，メソッドの処理内容を実装できるレベルで記述する．

4.1.3　物理データモデル定義

論理データモデルをもとに，より実装に近いデータモデルを作成し，物理データベースの設計，作成を行う．論理データモデルで表現していなかったテーブルとその属性について明確にする．また，逆正規化を行うことによって，同時更新をするテーブルについてのデッドロックの可能性や効率面などを考慮し，論理データモデルの構造を検証する．

物理データベースを設計・生成するためには，以下に挙げる作業を行う．

・効率面から物理データモデルの構造を検討する．
・データへアクセスする際の最適な経路を決定する．
・データ量やデータの更新頻度などを考慮し，各種領域のサイズを決定する．
・各種パラメータの設定を行う．
・データベースを生成する．

4.1.4　実装・単体テスト・結合テスト

クラス仕様を入力としオブジェクト指向プログラミングで実装する．単体テスト，結合テストまで実施する．

4.2　コンポーネント・システムの受入れ

統合テスト，システムテスト，ユーザテストなどシステム全体を稼働させるためのテスト計画を作成し，テストおよび検証を実施する．

4.2.1 テスト計画作成・テスト実施

テストの目的と実施範囲，スケジュールや体制，テスト環境などを明確にするために，テスト計画・テスト仕様を作成する．テスト計画・テスト仕様に基づいてテストを実施する．

各テストの内容を以下に記述する．テスト内容を比較するため，単体テスト，結合テストも記述しているが，結合テストまでの作業は 4.1.4 項で実施する．

1) 単体テスト

クラス単位や画面単位，共通部品単位などの実装単体レベルでの検証を実施する．実装した処理，処理分岐のパターンを全て網羅する．

2) 結合テスト

複数画面を操作して画面遷移を行い，ユースケース単位で動作検証を実施する．また，正常系，異常系の全てのシナリオの動作検証を実施する．他システムとのシステム間連携がある場合，連携の動作検証を実施する．現行システムからのデータ移行がある場合，移行データを使用した動作検証を実施する．

3) 統合テスト

日々の業務運用に従ってシステム全体の動作検証を実施する．具体的にはシステムが提供する複数の機能間や他システムとの連携，日替わりなどを繰り返すロングランテストを実施し，業務的観点からの全体整合性の検証を実施する．後述のユーザテスト（運用テスト）で使用する操作マニュアルはこの時点で作成する．

4) システムテスト

障害・復旧，負荷テスト，セキュリティなど，システム的な要求を満たすことの最終的な検証を実施する．

5) ユーザテスト（運用テスト）

エンドユーザの立場でシステム全体の動作検証を実施する．本番稼働直前の時期であり，エンドユーザの社内教育を実施することもある．

5. おわりに

本稿では，LUCINA for .NET のアーキテクチャと開発プロセスを適用した開発作業について紹介した．実際の開発現場では固有の状況（スケジュールや体制や制約など）が存在しており，LUCINA for .NET の開発プロセスをベースに独自の工夫を施すこともあるが，基本的な開発の流れは本稿に記した通りである．

オフショア開発では，各設計成果物の記述レベルを詳細化することは勿論，開発テンプレートをさらに厳密に規定し，受入れチェックリストを開発者に早期に周知・運用することで文化や体質の相違点を埋めていく作業が発生する．

オフショア開発における開発プロセスの周知や品質の確保に関する対策については，次の機会に譲ることにする．

＊1　アーキテクチャ：レスポンスやスループットに代表される性能，スケーラビリティ，可用性，保守性，信頼性，セキュリティに影響を与えるアプリケーション設計の主要要素．システム全体を特徴付ける要素技術やシステムの内部構造，システムを構成する部品，アプリケーションの設計方針．

＊2　コンポーネント・システム：業務データを外部から直接操作することを禁止し，業務データにアクセスするためのインタフェースをサービスとして外部に提供するもの，またはそれらサービスの集合体．業務データ（属性）と業務ロジック（操作）をまとめたグループに分割して情報隠蔽の単位とする．サブシステム相当の粒度．企業システム全体を構成するコンポーネント（部品）と捉え，コンポーネント・システムと呼ぶ．

執筆者紹介　白　木　和　彦（Kazuhiko Shiraki）

1991年日本ユニシス(株)入社．2200シリーズ上でのミドル・ウェアの開発を担当．地銀向け勘定系パッケージTRITONの開発を経て，2001年より商品先物取引基幹業務パッケージの開発に従事．現在，日本ユニシス・ソリューション(株)AD CoEコンピテンスセンタ.NET開発コンサルティングに所属．LUCINA for. NETを活用した開発標準策定サービスを担当．

幸　野　俊　介（Shunsuke Kono）

2003年日本ユニシス(株)入社．入社当時より.NETビジネス専任組織「.NETビジネスディベロプメント」に属し，.NETでのWebアプリケーションシステム開発，LUCINA for .NETを利用した標準化作業に携わる．現在，日本ユニシス・ソリューション(株)AD CoEコンピテンスセンタ.NET開発コンサルティングに所属．