LUCINA® for .NET 2005に基づく，人材モデルと育成指針

Skill Standard and Personnel Training Plans based on LUCINA® for .NET 2005

尾 島 良 司

要 約 LUCINA® for .NET 2005は，.NET Frameworkを用いたビジネス・アプリケーション開発のための，日本ユニシス独自の開発方法論である．この日本ユニシス独自という言葉は，日本ユニシスが作成したという意味だけにとどまらない．LUCINA for .NET 2005は，日本ユニシスの現状，おそらくは他のSIerの現状にも合致した開発方法論なのである．

現状に合致しているという言葉の意味は，社員全員が最新のコンピュータ理論を習得していなくてもよいということである．社員全員がオブジェクト指向を習得済みで，最新のプログラミング環境もビジネス環境もインフラストラクチャを構成するプロダクトも知っているのであれば，RUP(Rational Unified Process)を適用すればよい．しかし，現状はそうではない．

本稿では，LUCINA for .NET 2005の体制面における特徴（ビジネスとソフトウェアとシステム基盤という独立性が高い同格のチームに分割し，チームをアーキテクトと作業者で構成する），チームを構成するメンバ（本稿では，ビジネス・アーキテクトと設計者，ソフトウェア・アーキテクト，プログラマ）が担当するタスク，そしてその育成方法を述べる．

LUCINA for .NET 2005で独立性が高いチームに分割するのは，あらゆる分野を習得した超人の育成は困難であることと，技術の進化によって独立性を担保することが可能になったためである．チームを構成するメンバがアーキテクトと作業者に分かれているのは，資質を考慮すると，社員全員に高度な専門教育を施すのは効率面での無駄が大きいためである．

LUCINA for .NET 2005では，専門教育を受けたアーキテクトがそれぞれの専門分野でアプリケーション全体を抽象化し，作業者が具象化作業を担当する．その結果，大多数を占める作業者に対する高度な専門教育は不要となり，残る少数のアーキテクトに対しても専門分野の教育だけで済む．LUCINA for .NET 2005は，人材育成における費用対効果を大きく高め，現状の社員構成でビジネス・アプリケーション開発を成功へと導くのである．

Abstract LUCINA® for .NET 2005 is an original methodology of Nihon Unisys for business application development that uses .NET Framework. “Original” doesn't mean only “Made in Nihon Unisys”. LUCINA for .NET 2005 is suitable for the current state of Nihon Unisys (and perhaps other SIers).

“The current state” means that many employees don't know about latest computer theories. If all employees were familiar with object-oriented-*, products and business, we could choose RUP (Rational Unified Process). But maybe, the current state is not so.

This paper is written about LUCINA for .NET features and personnel training plans. LUCINA for .NET recommends three teams - Business, Software and System base, because there is no superman who knows everything. In each team, LUCINA for .NET recommends to draw a sharp line between architects and simple workers, because only a few people have a faculty of business, software and system base.

There is no need for the worker that is majority of employees to get professional education. The field of professional education for the architect is only architect's specialized field. As a result, LUCINA for .NET 2005 greatly improves cost-effectiveness in the personnel training.

1. はじめに

その対象がなんであれ，費用対効果は重要である．人材育成も然り．万能な人材を育成するための膨大な費用は許容できないし，現場で要求されない技術を教育しても無駄である．

だから，人材育成の場では開発方法論を活用してきた．開発方法論には，開発時の責務であるロールと，その責務を遂行する際に必要となる技法が含まれている．開発方法論を活用すれば，各ロールに適した最小数の人材を選抜し，必要とされる最小数の技法を教育するだけで済むのである．

しかし，このやり方が成果をあげているとは言いがたい．成果が上がらない理由は，世間一般の開発方法論が，コンピュータ・サイエンティストと呼ばれる特別優秀な人間によって，ソフトウェア・エンジニアリングの深い知識を持つ彼らと同類の人間を対象に作成されているものだからだと考える．実際に適用するには前提が特殊すぎるのだ．

たとえば，RUP（Rational Unified Process）では，UML（Unified Modeling Language）というモデリング言語を用いて設計する．たしかにUMLそのものは簡単であり，容易に習得できる．しかし，UMLはオブジェクト指向に立脚しているのだ．そして，日々オブジェクト指向言語を扱っているプログラマはともかく，大多数の設計者はオブジェクト指向を知らないのである．だから，RUPを教育するにはオブジェクト指向も教育しなければならない．これではRUP教育で成果をあげることが困難なのも当然であろう．

本論で取り上げるLUCINA® for .NET 2005は，.NET Frameworkを用いたビジネス・アプリケーション開発のための，日本ユニシス独自の開発方法論である．上で挙げた“前提が特殊すぎる”という問題とは無縁であり，人材育成においても十分な効果をあげることが可能だと考える．

以下，本論文では，LUCINA for .NET 2005が人材育成面で費用を下げて効果を高めていくメカニズムについて論じる．具体的には，LUCINA for .NET 2005が想定しているチーム分割と人材モデル，個々の人材モデルが担当する作業とその育成方法である．人材育成の指針作成の参考となり，人材育成の対象である現場社員のキャリア・プラン作成の参考になれば幸いである．

2. チーム分割とキャリア・パス

本章では，LUCINA for .NET 2005におけるチーム分割とキャリア・パスについて述べる．

LUCINA for .NET 2005のチーム分割における特徴は，ウォーター・フォールのようなタスクに基づく階層関係を持たないことである．LUCINA for .NET 2005ではプロジェクト・メンバをビジネス，ソフトウェア，システム基盤の三つのチームに分割するのだが，これらのチームは同格である．同格である以上，いわゆる設計者の報酬がそれ以外のチームのメンバよりも高かったりすることはない．設計者がプログラマに指示を出したり，プログラマが昇進して設計者になったりすることもないのである．

LUCINA for .NET 2005のキャリア・パス上の特徴は，アーキテクチャを作成するアーキテクトとその指示で働く作業者に分かれていることである．これだけでは設計者とプログラマの関係と同様に感じられるかもしれないが，同じ分野を対象とする点で明確に異なっている．職種が異なるのではなく，単純に能力が異なるのだ．LUCINA for .NET 2005のキャリア・パスは，作業者に始まり，同一の分野を担当する優秀な技術者，つまりアーキテクトへと進むこと

になる．

LUCINA for .NET 2005 がこのような体制とキャリア・パスを採用しているのは，人材育成上の理由による．本章では，その理由を「ウォーター・フォールの人材育成上の課題」とそれを解決するための「上下関係のない三つのチーム」，LUCINA for .NET 2005 の体制におけるキャリア・パスである「各種アーキテクトと作業者」として述べる．

2.1 ウォーター・フォールの人材育成上の課題

ウォーター・フォールでは，後に続く工程は前工程のアウトプットに基づく詳細化/変換作業である．例えば物理設計は論理設計書をシステム化が可能なレベルまで詳細化する作業であり，プログラミングとは物理設計書をソース・コードへと変換する作業である．このようなプロセス下では，担当する工程の知識に加えて，後に続く工程の知識も必要となってくる．

よって，ウォーター・フォールを採用する企業では，後に続く工程の知識を持った技術者を用意するために，ウォーター・フォールの工程とキャリア・パスを同一視してきた．プログラミング経験を十分に積んだ優秀な技術者を設計者に昇進させ，設計の経験を十分に積んだ優秀な技術者をアナリストに昇進させてきたのである．

このウォーター・フォールに基づくキャリア・パスは，技術や社会の変化が早い今日ではもはや成立しえない．オブジェクト指向やコンポーネント指向やサービス指向に日々触れているプログラマに向けた物理設計書を，COBOL の経験しか持たない設計者が書くのはナンセンスである．法律や商習慣が変化し続け，そして何よりシステム化対象領域が際限なく広がっている今日に，10 年前の古い小さなシステム設計の経験をもとに大きな新しいビジネスを分析することは不可能なのである．

それだけではない．ウォーター・フォールのキャリア・パスには，モチベーションを低下させるという弊害もある．ビジネス面での才能に溢れる新卒者であっても，まずは興味がないプログラミング修行をしなければならない．どれだけ優れたプログラミング能力を持つプログラマであっても，その扱いは単純労働者である．文書を書く日々が続き，ソフトウェア技術者としての能力が日々衰えていくことが確実だとしても，収入を増やすには設計者になるしか道は無いのである．

さらに，プロジェクトが失敗する可能性も高くなってくる．ウォーター・フォールのキャリア・パスでは，プログラミングは昇進前の賃金が安い人間の仕事であり，プログラマは社外からの供給に頼ることになる．自社社員のキャリア・パスは，設計者から開始である．プログラミングの知識があることを前提とするウォーター・フォールでの設計作業を，プログラミング経験を持たない人間が担当するようになってしまうのである．

2.2 上下関係のない三つのチーム

先に述べた課題は，分業可能な体制を構築し，それぞれの分野の独立性を高め，チームとすることで解決できる．プログラミングの知識を必要としない部分を抽出し，それぞれの分野の専門性を高め，専門のチームを割り当てていけばよいのである．

以上の考察をもとに，LUCINA for .NET 2005 では，“ ビジネス ” と “ システム基盤 ” をプログラミングから独立させた．また，プログラミングという言葉は単純労働の語感が色濃く残っているので，“ ソフトウェア ” と呼称するものとした．ビジネスの専門家は，機能要件を作

成し，作成されたシステムが機能要件を満たすかを検証する．システム基盤の専門家は，運用要件をまとめ，システム基盤であるインフラストラクチャを構築する．ソフトウェアの専門家は，非機能要件をまとめて，機能要件とあわせてインフラストラクチャに適合するようにプログラミングするのである．

この分業体制は，技術の進歩が可能にした．以前は言語やライブラリやツールが低機能だったために論理設計書とソース・コードに大きな溝が発生し，その溝を埋めるために物理設計書が必要であった．LUCINA for .NET 2005 が対象とする .NET Framework 2.0 と Visual Studio 2005 ならば，物理設計書の作成より低コストでのプログラミングが可能である．また，以前はインフラストラクチャが貧弱で，効率良く動作させるにはソース・コードから直接制御する必要があり，インフラストラクチャとプログラミングの両方の知識が必要であった．昨今の高性能/高機能なインフラストラクチャであれば，ソース・コードからの制御は不要である．

技術の進歩は専門家を不要にするとの意見もあるが，筆者は逆だと考える．技術が進歩して出来ることが増えれば，より専門的なビジネス分野が IT 化の対象となる．特化したビジネス分野に対応するには，ビジネスの専門家が必要なのである．インフラストラクチャが高機能化/高性能化すれば，更なる可用性や処理量が要求される．そのためには，プロダクトのすべてを知り尽くした専門家が必要なのである．そして，プログラミング環境が高度化しても，それを使いこなせなければ意味は無い．だから，最新のプログラミング技術を身につけたソフトウェアの専門家も必要なのだ．

LUCINA for .NET 2005 においては，これらの専門家は同格である．モチベーションを下げ，プロジェクトの失敗につながってしまう階層関係は一切ない．非機能要件，たとえば認証方式やセキュリティなどについては，ソフトウェアの専門家にその決定権がある．非機能要件を決定するには，プログラミングの専門知識が必要だからだ．逆もまた真である．ソフトウェアの専門家は機能要件に口出しをしない．ビジネスの専門知識を持たない門外漢では，ビジネスの専門家に対する指示を出すことは不可能だからである．

2.3 各種アーキテクトと作業者

しかし，LUCINA for .NET 2005 ではプロジェクト・メンバ全員が平等なわけではない．ビジネスである以上，報酬はその成果で決定されるべきだからだ．プログラミングと設計文書作成の間に報酬上の差異があってはならないが，より専門的なプログラミング，より価値の高い設計文書作成を担当する技術者には高い報酬と地位を与えるべきだと考えている．

ただし，特別な役割を持たない人間に高い報酬を支払うことは困難であろう．だから，LUCINA for .NET 2005 では，この役割として“抽象化によって導かれた全体像の作成”を用意した．この抽象化によって導かれた全体像とは，無矛盾で全体最適で，それを元に具象化が可能な成果物である．無矛盾で全体最適な成果物を作成するには，本質的でない部分を切り捨てて単純化しなければならない．これがいわゆる抽象化だ．そして，システム化対象の全てに言及していなければ，無矛盾や全体最適を証明できない．だから，全体像なのである．さらに，抽象化の過程で切り捨てたものを補うことが具象化である．「補う」だけなのだから，具象化は容易な作業である．

LUCINA for .NET 2005 では，この抽象化によって導かれた全体像を，ビジネスとソフトウェアとシステム基盤のそれぞれで作成する[*1]．抽象化とは本質ではない部分を切り捨てること

であるため，ビジネスとソフトウェアとシステム基盤で，それぞれが異なる本質に基づいて抽象化によって導かれた全体像を作ることは可能なのである．

詳細は 3 章と 4 章で述べることにして，ここでは用語を定義する．LUCINA for .NET では，先に述べた抽象化によって導かれた全体像のことを“アーキテクチャ”と呼ぶ．このアーキテクチャの作成を担当する技術者が“アーキテクト”である．LUCINA for .NET 2005 の開発プロセスには，“ビジネス・アーキテクト”と“ソフトウェア・アーキテクト”，“システム基盤アーキテクト”の 3 種類のアーキテクトが存在する．

アーキテクトではない技術者は，従来と同じ名前で呼ばれる．すなわち，“設計者”，“プログラマ”，“プロダクト・スペシャリスト”である．LUCINA for .NET 2005 は，設計者やプログラマ，プロダクト・スペシャリストには高度な技術力は要求しない．設計者やプログラマ，プロダクト・スペシャリストは，アーキテクトの指示に基づいて作業する作業者なのである．

LUCINA for .NET 2005 でのキャリア・パスは，設計者，プログラマ，プロダクト・スペシャリストのどれかを選択することから始まる．経験を積み，アーキテクチャを作成可能なほどに有能であれば，ビジネス・アーキテクト，ソフトウェア・アーキテクト，システム基盤アーキテクトへと昇進する．アーキテクトになっても担当する分野が変わらないのだから，ウォーター・フォールのキャリア・パスで発生したような課題は発生しないのである．

ここまでを図 1 にまとめた．各チームは同格であること，そしてチーム内はアーキテクトと作業者の階層関係にあることに注意していただきたい．アーキテクトと作業者が担当する具体的な作業内容については，3 章および 4 章で育成方法と併せて述べる．なお，本稿では，LUCINA for .NET 2005 の対象外であるプロジェクト・マネージャとシステム基盤アーキテクト，プロダクト・スペシャリストについては割愛する．

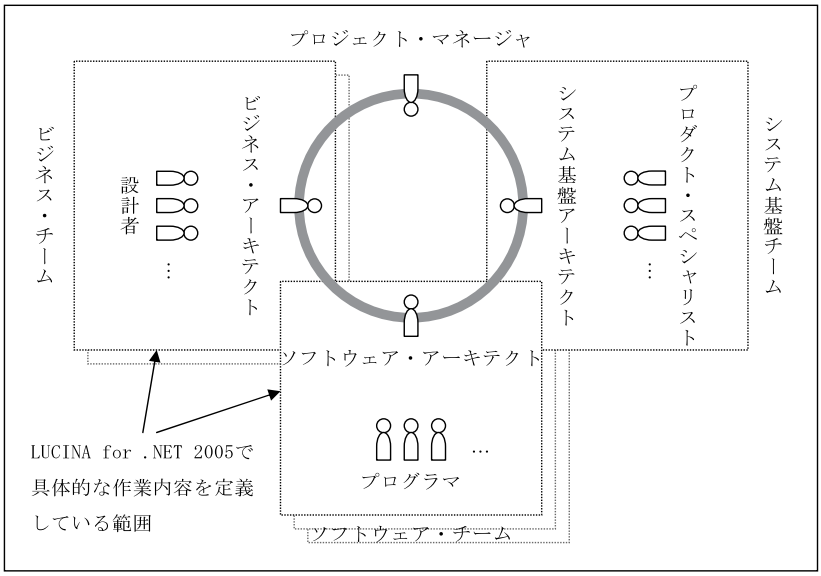

図 1　LUCINA for .NET 2005 におけるプロジェクト体制

3.　ビジネス・アーキテクトと設計者

本章では，ビジネス・アーキテクトと設計者が担当する作業について述べ，ビジネス・アー

キテクトと設計者を育成するための施策を述べる．

LUCINA for .NET 2005でビジネス・アーキテクトが作成する成果物は，ER（Entity Relationship）図である．ER図は，ビジネス・アプリケーションの主たる関心事であるRDBMS（Relational Database Management System）を通じてシステムの全体像を漏れなく表現可能で，一目で理解できるほどに抽象的である．

設計者は，ER図を基にしてロジックや画面を定義する．LUCINA for .NET 2005の設計面での特徴は，ER図をその中心に置くことである．設計者は，独力で構造化するのではなく，エンタープライズ・アプリケーション・アーキテクチャ・パターンのTable Moduleに従い，ER図のエンティティにメソッドを追加する形で構造化していく．画面定義も，ER図を基にして作成するのである．

LUCINA for .NET 2005がこのような作業分担を採用しているのは，優秀な技術者であるビジネス・アーキテクトの成果を可能な限り広く活用した方がリスクを削減できるからである．本章では，以降の「ビジネス・アーキテクト：ER図で全体設計」でビジネス・アーキテクトの担当する作業を，「設計者：Table Moduleパターンで機械的に構造化」で設計者が担当する作業を述べる．その上で，「ビジネス・アーキテクトと設計者の育成」として人材育成のための施策を述べる．

3.1 ビジネス・アーキテクト：ER図で全体設計

ビジネス・アーキテクトは，いわゆるモデリングを担当する．モデリングとは，特定の記法を用いて対象を分析することである．モデリングの成果物であるモデルは，抽象化によって導かれた全体像，つまりLUCINA for .NET 2005におけるアーキテクチャなのである．

記法の選択の対象になるのは，2006年現在ではUMLとER図に絞られる．そしてLUCINA for .NET 2005は，上記二つの記法からER図を選択した．選択の理由は，以下に述べる三つである．

1） ビジネス・アプリケーションのほとんどはデータを登録/検索するアプリケーションであり，その本質をデータ構造で表現できるものが多い．
2） UMLを教育するにはオブジェクト指向の教育が必要であり，コストがかかる．
3） UMLはデータとロジックの両方を表現できるので，設計の最後まで使用できてしまう．貴重なビジネス・アーキテクトに，設計のすべてをやらせるべきではない．

つまり，費用対効果である．UMLは高機能だが高価であり，ER図は安価な上に必要十分な効果があると判断したのである．ビジネス・アーキテクトという高価なリソースを投入して詳細化作業をさせるより，より安価な設計者に詳細化作業を担当させるべきだという判断でもある．

ER図の表現可能な範囲が狭いという点を，LUCINA for .NET 2005はメリットとして捉えているのである．たしかにUMLの表現力の高さは魅力的である．優秀な技術者にUMLを与えれば，アプリケーションの詳細部分でも，あとはオフショアに出すだけというレベルまで詳細に記述可能であろう．

しかし，優秀な技術者は貴重なのである．彼らがすべてを担当するのは不経済だ．しかも，

優秀な技術者はすべてを自分で作業することを好む．だから，道具のレベルで制約を作り，他の技術者に作業を引き継がせるべきだと考える．データ構造しか表現できない ER 図を使用すれば，データ構造の作成より後の作業を強制的に引き継がせることが可能となるのである．

また，UML を記述するには，プログラミングの知識が必要になる．UML ではメソッドについて記述できるためである．ビジネスの専門家であるビジネス・アーキテクトは，プログラミングよりもビジネスを学ぶべきだ．その意味でも，表現可能な範囲が狭い ER 図の方が望ましいと考える．

以上の理由により，LUCINA for .NET 2005 は，あえて UML よりも表現力が劣る ER 図を採用したのである．

3.2 設計者：Table Module パターンで機械的に構造化

ER 図はデータの構造しか表現できないため，ロジック部分を後から追加しなければならない．LUCINA for .NET 2005 では，設計者がこの追加作業を担当することになる．

ロジックの設計において重要なことは，重複を無くすことである．共通する部分を抜き出して再利用可能なコンポーネントにまとめ，再利用可能なコンポーネントを活用する形で新たなロジックを設計する．重複をなくせば生産性と保守性が高まり，SI ビジネスの採算がとれるようになり，保守コストも低減するのである．

しかし，重複をなくすロジックの構造を一から作っていくには，アプリケーション全体のロジックを理解しなければならないという問題がある．何処に何があるのか知らなければ，重複部分を抽出することはできないためである．さらに，プログラミングの知識が必要だという問題もある．プログラミングの知識がなければ，デザイン・パターンや言語機能，ライブラリが提供する再利用可能コンポーネントを活用することができないのである．

作業者である設計者に，アプリケーションに含まれるすべてのロジックを理解して記憶することを要求するのは無理がある．なにしろ，現在のアプリケーションは巨大なのだ．一つの小さな業務を省力化すればよかった時代の考え方は通用しない．また，設計者にプログラミングの知識を求めるのも非効率である．設計者は将来のビジネス・アーキテクトなのだから，プログラミングよりもビジネスを勉強すべきである．

以上の考察から，設計者が担当する作業に対する二つの条件が導ける．一つ目は，重複する部分を含むロジックが小さな範囲に収まることである．重複を探す範囲が小さければ，設計者でも重複を発見して排除することが可能になる．二つ目は，プログラミングの知識が不要なことである．設計者には，ビジネスの知識を求めるべきなのだ．

まずは，重複する部分を含むロジックを小さな範囲に収める方法を考察しよう．幸いなことに，ER 図がある場合には，重複を含むロジックを機械的な作業で一か所にまとめることが十分に可能である．エンタープライズ・アプリケーション・アーキテクチャの Table Module パターン[1]を適用すればよいのだ．

Table Module パターンとは，テーブルを仮想的なクラスと考え，そこにメソッドを配置していく設計方式である．他のテーブルのデータを扱うロジックを定義する場合は，他のテーブルにメソッドを追加し，その追加したメソッドを呼ぶ形でロジックを定義することになる．この方式ならば，同じデータを扱う，つまりは似たロジックが一か所にまとまることになる．しかも，機械的な作業によってである．この方式なら，設計者であっても十分に重複を排除でき

るだろう．

次に，プログラミングの知識を不要にする方法を考察しよう．それには，以下に示す二つが有効である．一つ目は，非機能要件の排除である．非機能要件，性能や認証，トランザクション制御方式などは，すべてソフトウェアの専門家に任せるべきである．二つ目は，外部仕様までしか担当させないことである．アプリケーションは最終的にはオブジェクト指向で作成され，そのオブジェクト指向の最大のメリットはインタフェースと実装の分離である．だから，そのインタフェースまでを設計者が定め，実装部分はソフトウェアの専門家がデザイン・パターンや言語機能，ライブラリの機能を活用して作成すればよいのだ．

詳細は割愛するが，ER 図から画面定義を抽出することも可能である．この逆の，そして広く使われている，ER 図より前に画面を定義する方式は絶対に避けなければならない．ER 図は一目で全体を把握できるが，複数の画面定義を一目で把握することはできないからである．画面定義を記憶し，他の画面定義と突き合わせて重複や矛盾を解消するという，設計者の能力を超えた作業が必要となってしまうのである．

もちろん，ここに述べた設計者が担当する作業のすべては，ER 図が正しいことをその前提としている．LUCINA for .NET 2005 がこのような前提をおけるのは，ER 図の作成を能力が高いビジネス・アーキテクトに任せているためである．

3.3 ビジネス・アーキテクトと設計者の育成

近代的なアプリケーション構造を簡略化して示すと，図 2 のようになる．つまり，アプリケーションは，画面とサービス，データベースで構成される．このうち，網掛けした部分（画面のインタフェースとサービスのインタフェースとデータベース）がビジネス・アーキテクトと設計者の担当範囲である．画面のインタフェースとは画面定義であり，サービスのインタフェースとはメソッドの外部仕様であり，データベースとは ER 図および実際のデータである．画面の中とサービスの中，いわゆる物理設計の部分まで含む範囲は，ソフトウェア・アーキテクトとプログラマが担当する．

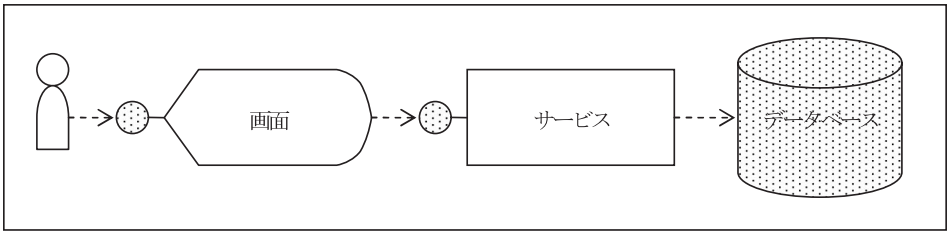

図 2　アプリケーションの全体像

ビジネス・アーキテクトと設計者の両方に必要とされるのは，対象分野の業務知識である．ヒアリングを通じて業務を勉強するのでは，設計が終わるまでにとても長い時間がかかってしまう．顧客の要求を咀嚼せずに取り入れていては，使われない機能を満載した無駄に大きいアプリケーションが完成してしまう．顧客が提出した矛盾を含む画面定義を基に ER 図を描くのでは，矛盾を解消するための大量のプログラミングが必要となってしまう．これでは SI ビジネスには成立しない．だから，豊富な業務知識をもとに ER 図のひな型を描き，業務知識を活

用して ER 図から画面やサービスを導いて顧客を説得すべきなのである．

そのためには，ビジネスの教育コースの開催が有効である．現在，日本ユニシスの教育コースには，コンピュータ技術についてのものが多い．しかし，業務知識を身につけることが必要なビジネス・アーキテクトと設計者にとっては，UML や .NET　Framework 等のコンピュータ技術の教育コースはあまり重要ではない．対象となる顧客のセグメントで実施されているのと同等のビジネス教育をしなければならないのである．

ビジネス・アーキテクトには，業務知識に加えて抽象化の能力が必要である．ER 図の作成は抽象化であり，抽象化は特殊な能力である．方程式を適用して解を導くのではなく，様々な事象から方程式を導く能力なのである．

だから，抽象化の能力が高い社員を選別することが重要である．現在のところ，抽象化の能力を持った社員は，データベース・スペシャリストに多いように見える．まずは，データベース・スペシャリストの中からビジネスに興味がある人間を募り，業務知識を教育し，ビジネス・アーキテクトとすべきだと考える．それと並行して，LOB（Line of Business）に所属する人間から抽象化能力が高い人間を選別する．ロジカル・シンキングを教育して選別し，ER 図の記法を教えて選別し，様々な ER 図を描かせて選別する．選別に残る者は少ないかもしれないが，LUCINA for .NET 2005 ではそれほど大量のビジネス・アーキテクトを必要としないので問題とはならないだろう．

4.　ソフトウェア・アーキテクトとプログラマ

本章では，ソフトウェア・アーキテクトとプログラマが担当する作業について述べ，ソフトウェア・アーキテクトとプログラマを育成するための施策を述べる．

LUCINA for .NET 2005 でソフトウェア・アーキテクトが作成する成果物は，ソフトウェア・アーキテクチャである．ソフトウェア・アーキテクチャとは，アプリケーションを代表する機能をすべて実装したソース・コード（“ アーキテクチャ・プロトタイプ ” と呼ぶ）とそのソース・コードと同じ形で残りの機能を実装するための手引書（“ ソフトウェア・アーキテクチャ説明書 ” と呼ぶ）である．いわゆる開発標準に相当するのだが，主はアーキテクチャ・プロトタイプというソース・コードである点に注意していただきたい．

プログラマは，ビジネス・アーキテクトが作成した ER 図と設計者が作成した外部仕様を，ソフトウェア・アーキテクトがソフトウェア・アーキテクチャで指示した通りにプログラミングする．内部仕様作成のための特別なフェーズは準備せず，ソフトウェア・アーキテクチャと事後の設計の洗練である “ リファクタリング ” で代替する．

LUCINA for .NET 2005 がこのような体制を採用しているのは，物理設計フェーズ（内部仕様書を作成する）に相当する作業の大部分をソフトウェア・アーキテクチャで代替可能なこと，数が多いプログラマの人件費を抑えれば採算上有利なこと，物理設計フェーズに含まれる作業はソフトウェア・アーキテクトやプログラマがその専門性を生かして実行すべきという判断の結果である．本章では，「ソフトウェア・アーキテクト：アーキテクチャ中心で物理設計を簡略化」と「プログラマ：リファクタリングで物理設計の残りを代替」でそれぞれが担当する作業を述べる．その上で，「ソフトウェア・アーキテクトとプログラマの育成」として人材育成のための施策を述べる．

4.1 ソフトウェア・アーキテクト：アーキテクチャ中心で物理設計を簡略化

ソフトウェア・アーキテクチャと聞くとUMLで作成したモデル図を想像する場合が多いだろう．しかし，LUCINA for .NET 2005におけるソフトウェア・アーキテクチャの本質はソース・コードである．ソフトウェア・アーキテクトは，アプリケーションを代表する機能を選択し，その機能をリリースが可能なレベル，すなわちテストが完全に終わって顧客が実際に使用可能なレベルまで実装する．このソース・コードをアーキテクチャ・プロトタイプと呼ぶ．

ソフトウェアにおける，抽象化で導かれた全体像とは，機能の相違を無視してソフトウェア構造を導くことだと定義できる．なぜなら，ビジネス・アプリケーションでは，機能が異なる場合でもソース・コードの構造は同じだからである．たとえば，どの機能もプレゼンテーション・レイヤとビジネス・レイヤに分割され，ビジネス・レイヤはビジネス・ロジック・コンポーネントとデータ・アクセス・オブジェクトとビジネス・エンティティで構成され，アプリケーション全体で同じトランザクション制御方式が採用される．よって，どの機能にも当てはまるソフトウェア構造を導くことこそが，アプリケーション全体の定義だといえるのだ．

このソフトウェア構造には，非機能要件を満たしていなければならないという条件が付く．そして，非機能要件を満たすことを検証するためには，コンパイルすればテスト可能なソース・コードで表現されている必要がある．どの機能でもソフトウェア構造は同じなので，ソフトウェア・アーキテクトは適当な機能をアーキテクチャ・プロトタイプとして実装すれば，負荷テストまで含めた完全なテストを実施できるのである．

そして，アーキテクチャ・プロトタイプは，プログラマが実装する際の手本となる．その際には，ソース・コードに込めた意図をプログラマが完全に理解できるとは限らないので，アーキテクチャ・プロトタイプを説明する文書が必要になる．これがソフトウェア・アーキテクチャ説明書であり，プログラマとソフトウェア・アーキテクトの共通言語であるUMLを活用して記述される．

さらに，ソフトウェア・アーキテクトは，非機能要件を定義する役割も持つ．ビジネス・アーキテクトや設計者が非機能要件というプログラミング上の話を決定するのは不適切だからである．だから，ソフトウェア・アーキテクトが顧客と非機能要件を打ち合わせることになる．LUCINA for .NET 2005で言う非機能要件とは，性能や可用性だけではない．認証方式やエラー・リカバリの方式，他システムとの接続方式などのソフトウェア化において考慮しなければならない点すべてを含むのである．

4.2 プログラマ：リファクタリングで物理設計の残りを代替

ここまでを整理する．ビジネス・アーキテクトによって，すでにデータベースは作成されている．設計者によって，作成するメソッドの外部仕様は定義済みである．そして，ソフトウェア・アーキテクトによって，プログラミングの方針はすべて定められている．

必要なものは揃っているのだから，あとは，プログラマがプログラミングすればよい．ここまでで，動作するビジネス・アプリケーションが出来上がる．

しかし，ビジネス・アプリケーションは，動作するだけでは不十分なのである．将来の保守が容易な，洗練された設計がなされていなければならないのだ．だから，LUCINA for .NET 2005は，動作するソース・コードが出来上がったら“リファクタリング”することを要求している．

リファクタリングとは，ソース・コードの重複する部分をメソッドとして抽出したり，クラス構造を変更したりして設計を洗練していくことである．事前に完璧な物理設計をするのではなく，ある程度の設計をもとに(美しくなくてもよいので)動作するソース・コードを作成し，その後に動作を保ったままで設計を洗練していくのである．

この説明だけでは，非常に危険な行為だと感じるかもしれない．しかし，リファクタリングはその実施方法が手順として定義されているのである．長くて保守性と再利用性が低いメソッドを分割する方法，ループが深くなった場合のメソッドによる置き換えの方法，クラスを分割したり，逆にクラスを統合したりする方法．これらが詳細な手順としてパターン・ランゲージの形で定義されている．だから，リファクタリングの本[2]を用意するだけで，プログラマが安全にリファクタリングすることは可能なのである．

4.3 ソフトウェア・アーキテクトとプログラマの育成

ソフトウェア・アーキテクトにもプログラマにも，プログラミング能力が必要である．この能力を身につけるための訓練は簡単だ．プログラミング能力とは洗練された美しいソース・コードを作成する能力なので，設計を洗練してソース・コードを美しくするリファクタリングを日々実施していくだけでよい．日々の業務にプログラミングが含まれているので，その能力が衰えることもない．

だから，わざわざ能力を高めるための教育コースを開催しなくてもよい．ソフトウェア・アーキテクトによる OJT（On Job Training）で十分である．プロジェクトに配置してプログラミングの機会を失わないようにし，ソフトウェア・アーキテクトという優秀な技術者を身近に配置するだけでよい．

以上の考察から，プログラマに対しては，プログラミングの基本とリファクタリングの一般的な知識と UML の読み方を教育するだけで十分であると結論できる．しかし，ソフトウェア・アーキテクトの場合は，そう簡単ではない．

まず，要素技術に対する深い知識と理解が必要である．そのためには，要素技術を学習するための機会と時間を与え，すでに要素技術に対する見解を持っているソフトウェア・アーキテクトと要素技術の適用方針について議論する機会を与える必要がある．要素技術を学習する機会としては，.NET　Framework のベンダである Microsoft のコンサルティング・サービスが優れたワークショップを開催しているので，それに参加させればよい．ソフトウェア・アーキテクトとの議論は，議論するための場所と工数を確保するだけで十分だと考える．

次に，顧客とのコミュニケーション能力が必要である．プログラミングの能力が高い技術者はコミュニケーション能力が低い傾向にあるので，ロール・プレイを含む実践形式のセミナーに参加させる．そして，ビジネスなどの無関係な，ただし初心者向けのセミナーへも参加させる．後者のセミナーへの参加の目的は，知識のレベルが異なる（だからと言って知能のレベルが異なるわけではない）他者とのコミュニケーション方法を学ぶことである．知識がない側を経験し，知識がない者相手の説明を目にすることは，ソフトウェア・アーキテクトになる大きな助けとなるだろう．

5. お わ り に

筆者は，LUCINA for .NET 2005 を活用することで，人材育成の費用対効果を高めることが

できると確信している．

多数を占める設計者とプログラマ，そしてプロダクト・スペシャリストが担当する作業の技術レベルを低く設定しているので，これら作業者の育成費用は大きく下がる．少数のビジネス・アーキテクトとソフトウェア・アーキテクト，そしてシステム基盤アーキテクトでまかなう仕組みになっているので，アーキテクト一人当たりの費用はさておき，全体での育成費用は安価になる．キャリア・パスを進んでも必要とされる知識が変わることはなく，それまでと同じ作業も担当するので，キャリアの境界での一からの教育が不要で，かつ腕が落ちる心配もない．そして，優秀な技術者である各種アーキテクトがそれぞれの視点でアプリケーション全体をカバーするので，品質と生産性が向上して採算が大きくプラスになる可能性が高くなるのである．

＊1　ただし，LUCINA for .NET 2005では，システム基盤については全体像を定めることしか定義されておらず，その詳細は定義していない．これは，LUCINA for .NET 2005の対象がアプリケーションの実装までであるためである．

参考文献　［1］　マーチン・ファウラー（著），長瀬嘉秀（訳），株式会社テクノロジックアート（訳），「エンタープライズアプリケーションアーキテクチャパターン」，翔泳社，2005年4月，P 133〜141

［2］　マーチン・ファウラー（著），児玉公信（訳），平澤章（訳），友野晶夫（訳），梅沢真史（訳），「リファクタリング−プログラムの体質改善テクニック」，ピアソンエデュケーション，2000年5月

執筆者紹介　尾　島　良　司（Ryoji Ojima）

1993年日本ユニシス(株)入社．PowerBuilder，C++によるオープン・システムの開発を経て，1996年にJavaの評価と社内への普及を担うJavaプロジェクトに参加．2002年より.NETビジネスディベロプメントにて.NET Frameworkを活用したビジネスの創出に従事．開発方法論であるLUCINA for .NETを策定している．