国際競争時代のコスト構造改革と需要拡大を支える航空基幹システムの世代交代「AirCore」

AirCore, New Generation of Airline Core System supporting Cost Reform and Demand Expansion in Age of Global Competition

佐　藤　　覚，小山田　和人，平　松　敦　郎

要　約　航空業界で中核をなす基幹システムとして，航空予約システムが挙げられる．大手航空会社にて使用されている航空予約システムの大部分は，1960年代から1970年代にかけてメイン・フレーム上で開発されたものであり，長年の機能拡張により，システムが複雑化・肥大化している．また，使用言語がAssemblerまたはFORTRAN主体のため，開発，保守を行う技術者の確保が困難でコスト高となっている．一方で航空会社にとってこのシステムは重要な戦略システムであり，市場のニーズに迅速に対応するため，短期開発と間断の無いIT投資が必要とされている．米国Unisys社（以降Unisys）はこの問題を解決するためオープン・アーキテクチャをベースとしたAirCoreの開発に着手した．AirCoreは，コスト削減，短期開発を可能とすると共に，より戦略的な顧客中心のシステム基盤を提供する．また，サービス指向アーキテクチャに基づきアプリケーションが明確にモジュール化されており，他のモジュールに対する依存度を最小限にすることにより開発の容易性，保守性を確保している．さらに開発手法として，Unisysの開発技法URUP（Unisys Rational Unified Process）を採用しており，開発リスクの軽減と品質維持を図っている．AirCoreの開発は，汎用機で培った基幹業務系開発運用ノウハウを活かしたURUPをベースにとした開発手順に構成管理とプロジェクト管理が組み込まれているのが特徴である．本稿はこれらを簡単に紹介する．

Abstract　The airline passenger reservation system is one of the most important core systems in the airline industry. The most of passenger reservation systems used at major airlines were developed on the main frame system in 1960's and 1970's, and many years of function enhancement have made the reservation system complicated and enlarged. In addition, the use of assembly language or FORTRAN as the system development language causes the difficulty to secure the development and maintenance personnel, and the increased costs. On the other hand, the reservation system needs the endless short-term development with the continuous IT investment to respond quickly to market needs. Unisys Corporation (hereinafter Unisys) has launched the development of AirCore, a new solution package of airline system based on the open architecture. AirCore enables not only the reduced costs and the short-term development but also the provision of the strategic solution with customer-centric features. AirCore is composed of the business components based on SOA. And Unisys uses URUP (Unisys Rational Unified Process) as the development methodology of AirCore to reduce the development risks and to keep the quality assurance. The development methodology of AirCore has the feature of discipline with the configuration management and the project management based on URUP.

1. はじめに

航空業界では，ローコスト・エアラインの台頭を始めとした近年の価格競争激化に加えて，原油価格の高騰等，厳しい経営環境が継続しておりグローバルでの統廃合，コスト構造改革が進んでいる．米国では，2005 年に入ってデルタ航空とノースウェスト航空が連邦破産法 11 条（日本の民事再生法に相当）を申請し，現在上位 7 社のうち実に 4 社が同法の適用状態というのが現状である．この中にあって航空各社では「IT コスト削減」が大きな命題の一つとなっている．一方で，メガキャリアといわれるネットワーク・エアラインでは時代に応じた顧客サービスで他社への優位性を維持することが常に求められており，そのための IT 投資を間断なく実施している．これら相反する要求を満たすため Unisys は，オープン技術を用いたシステム基盤の世代交代に取り組んでいる．本稿は，Unisys が開発中の航空業界向け次世代旅客系ソリューション・パッケージ AirCore の概要とそのアーキテクチャ，そしてオープン技術上にそれらを構築する手法がこの業界変革にどのような対応を提案しているかを紹介する．

2. 航空業界を支える情報基盤の動向

航空業界の IT 投資で戦略的な観点で最も重要視されているのが航空予約システムである．近年のインターネットによる顧客への直接販売も裏で支える予約システム無しには実現できない．また，提供する機能も予約システムの制限に縛られているのが実情である．この航空予約システムは歴史的にも古く，最初の航空予約システムは，1960 年代にオンライン・リアルタイム・システムとして，IBM のメイン・フレームを使用して稼働した．その後，1970 年代後半に Unisys（当時，UNIVAC）のメイン・フレームにて稼働する USAS（Unisys Standard Airline System）が開発された．Unisys は，航空予約システムである USAS の提供を初めとして，200 社以上の航空会社（上位 25 社のうち 22 社），100 箇所以上の空港にサービスを提供しており，全世界の搭乗者の 29% が Unisys のシステムを利用している．本章では，航空予約システムに関連する動向と，40 年以上に亘りメイン・フレームにて稼働している，航空予約システムの課題を提示する．

2.1 航空予約システムの動向

1） 自社保有から外部サービス利用へ

航空予約業務は，旅客に対応する予解約，発券，搭乗とインベントリとよばれる在庫（座席）を管理する 4 つの基本業務に分類される．当初，全業務を処理するシステムを，各航空会社が独自に保有していた．しかし，世界規模で事業を展開する航空会社にとって，独自に端末，ネットワークを展開，維持するコスト負荷は，許容できる範囲を超えるものとなった．

この問題を解決するため，予解約，発券に係るシステムを航空会社から分離し，ASP として各航空会社に機能を提供する GDS（Global Distribution System）が設立された．GDS は，端末，ネットワークを展開し，予解約，発券業務のフロント・エンドとして機能する．各航空会社は，搭乗者（設立当初は，予約）あたりの手数料を支払うことにより予解約，発券のシステムを保有する必要がなくなり，搭乗およびインベントリ管理のみを行えば良いこととなって，コスト削減が実現された．

2） ローコスト・エアラインの出現

1980 年代に入ると，米国の規制緩和政策を受け，既存のエアラインと異なるビジネス・モデルとして，低額の航空運賃を提供するローコスト・エアラインが出現してきた．ローコスト・エアラインは，あらゆる面でのコストを削減（機内サービス，機種の統一など）することで低価格を実現しており，予約システムに係るコストも削減の対象としている．インターネットのみでの販売，予約変更の制限，他社との連携を行わないなどの方式を採用して GDS への支払いを不要とし，かつ自社システムとしての予約システムに必要となる機能を削減することで，低コストでこの要求を実現している．

3) 情報系の活用

規制緩和に伴い，自由に複数の運賃の設定，運航区間の設定などが可能となり，最適なインベントリ管理が求められることとなった．規制緩和以前は，いかに多くの乗客を搭乗させるかがインベントリ管理の主目的であったが，規制緩和以降は，運賃ごとの最適な座席配分により，収益の最大化を追求する必要が出てきた．予約システムにより作成された大量の情報を蓄積・分析する情報系の役割が大きく求められることとなった．また，蓄積された情報を科学的な手法を用いて分析し，その結果を予約システムのインベントリ管理に反映させる仕組みも必要である．

4) 他社，他業種との連携

旅客運送を行うには，自社の運航する便のみではなく，他航空会社の便を利用する必要がある．航空業界では，古くから標準化が行われており，業界団体により航空会社間の情報交換手順が定められている．また，旅客に対するサービスとして，ホテル，レンタカーなど旅行に必要となる各種素材の予約機能も提供されている．現在では，旅行代理店を始め，他社システムあるいは他業種システムとの間で，リアル・タイム接続が行われている．当初，業界独自の手順による接続が行われてきたが，接続先の拡大に伴い，EDIFACT あるいは XML が採用されるようになった．

5) 最新 IT の利用

インターネット，携帯電話，E メール，IC カード，XML などの最新 IT は，航空業界においても避けて通ることのできない技術である．また，従来，紙を媒体として使用していた航空券も，E チケット化により，電子情報として蓄積，交換が行われる様になってきた．これら技術の活用は，独立したシステムとして行われるのではなく，メイン・フレームの予約システムに対する入出力手段として利用されている．

6) 自社保有への回帰

インターネットの発展に伴い，個人が直接，航空予約システムへアクセスすることが可能になった．この環境変化は，航空会社にとって，外部サービス提供者への手数料を支払う必要のないことを意味し，再び，航空予約システムを自社保有することの是非が検討され始めてきた．但し，旧来型の形態ではなく，共同保有，アウトソーシング，オープン・プラットフォームの活用などコスト削減が可能となる方策が求められている．

2.2 航空予約システムの抱える課題

1999 年に 119 社の航空会社役員に対し実施された重要課題に関する調査[1]において提示された，財務上の課題には以下がある．

① 収益の改善（72%）

② コストの削減（62%）

③ 路線の最適化（59%）

現在においても，これら課題は継続して存在しており，航空予約システムの観点で，以下の課題として認識される．

1） システムの肥大化

現在，多くの予約システムはメイン・フレーム上で稼働しているが，それを取り巻くシステム群および他社・他業種との接続を行うため，オープン・プラットフォームが使用されている．航空業界内での標準化は行われているものの，旅客サービスの観点から，旅行関連以外の業界との連携も行われ，接続先ごとの固有な手順が必要となり，それらはゲートウェイ機能を持つサブシステムにより行われている．これらの要因から，多くの航空予約システムは，中央のメイン・フレームとそれらを取り巻く数百のサブシステムから構成されており，その全容を把握することすら困難な状況にある．

2） 複雑化と技術者の確保

現在の予約システムの基礎は，1960〜1970 年代に構築されたものであり，その後の規模拡大に伴い，数多くの変更が行われてきた．基本となるプログラムをベースに，機能拡張が行われ，いわゆるスパゲッティ・プログラムとなっている現実は否めない．

また，これらのシステムは主にアッセンブラか FORTRAN で記述されているため，開発，保守を行う技術者の確保が困難な状況にあり，かつ高額のコストが必要とされている．

さらに，システムの肥大化に伴い，各技術者の担当する領域が細分化され，システム全体あるいは特定の業務全体を把握する技術者が少なくなっており，いわゆる 2007 年問題は航空業界においても，懸念される課題となっている．

3） 短期開発の要求

航空予約システムは，航空会社にとって重要な戦略システムであり，市場のニーズに迅速に対応することが求められている．特に，急速に発展する IT の進歩への追従は，メイン・フレームを中心とする航空予約システムにとって，最も不得手とする領域であり，早急な改善が必要である．

4） コスト削減

上述のとおり，航空予約システムに掛かるコストは，上昇する要因を多く抱えており，他社との競争を行う上での阻害要因となっている．外部サービスの活用により，この問題を解決したかに見えた航空会社であったが，サービスを提供する側の GDS にとってもこれらの課題は同様であり，IT 関連コストの上昇は，搭乗者あたりの手数料に転嫁され，結果として，航空会社のコスト増加を引き起こしている．

また，従来の航空予約システムは，キャラクタ・ベースのインタフェース（CUI）を基本としており，端末オペレータには業務を遂行する上で必要となる全ての入力を習得することが求められる．これにより，端末オペレータに対する教育には期間が必要となり，コスト削減を阻害する要因の一つとなっている．

5） オープン化

長年培ってきたメイン・フレーム上の開発，運用ノウハウはシステムの安定稼働を実現してきた．反面，近年の IT 技術進歩の恩恵を得られていない現状もある．先進の IT 技術を取り込みつつ，メイン・フレーム並みの安定稼働をオープン・システムで実現するこ

とが求められている．

2.3　Unisys の取組み―世代交代へのアプローチ

前節の課題を解決するためにはこれまで行ってきた対処療法では限界があると判断した Unisys は，現有システムの改良/改善あるいは外付けのサブシステムによる対応を行うのではなく，AirCore というまったく新しいシステムを，最新技術を取り入れて再構築するアプローチを取っている．

その特徴は以下のとおりである．

1)　顧客中心の業務機能の実現

従来の航空予約システムは，係員の操作中心のシステムであり，戦略システムとして成長する際に後付けで顧客サービス機能を追加してきているが，AirCore は，完全にリメイクすることにより顧客データをシステムの中心に位置づけ，シームレスな顧客サービス機能を実現している．

2)　J 2 EE 採用によるオープン・プラットフォーム化（プラットフォーム・フリー）

J 2 EE で構築することによりオープン・プラットフォーム化を図っている．H/W や，OS，J 2 EE コンテナ，RDBMS の選択が可能でより柔軟なシステム構成が可能である．ただし，現在 AirCore は，J 2 EE コンテナとして Weblogic Server，RDBMS として Oracle を推奨している．

3)　完全な Web ベースのアプリケーション

標準的なインタフェースとして Web 技術を採用し，Web サービスを提供している．

4)　最新技術に基づいたモジュール構造化

サービス指向アーキテクチャに基づく業務のコンポーネント化とアプリケーションのモジュール構造化を採用し，アプリケーションが縦（業務によるコンポーネント化）と横（レイヤによるサービス分割）に明確に分離されている．加えて業務のコンポーネント化では，Unisys がこれまで培ってきた業務ノウハウが十分に生かされており，業務機能の最適な配置が実現されている．

5)　基幹業務としての運用に耐えうるシステム

メイン・フレームと同等の応答性，24 時間 365 日の連続稼働に耐えうる信頼性，オープン・システム利用のメリットを引き出す拡張性を備えている．

これらについては，3 章で AirCore のアーキテクチャ的な特徴，そして 4 章で AirCore を現実的なシステムとして構築するために利用したプロセス的な特徴について具体的に記述する．

3.　AirCore 概要

AirCore は，航空業務の中核をなす予約，発券，搭乗業務を対象とした大規模ミッションクリティカル・システムである．そこでは，40 年以上に亘り主流を成してきたメイン・フレームによるオンライン・リアルタイム・システムに取って代わるため，先端技術を採用している．「AirCore の目指すもの」すなわち AirCore の目標と先端技術を用いたその実現方法は以下のとおりである．

3.1 AirCore が目指すもの

表 1 は，AirCore の目標とその実現方法を簡単にまとめたものである．これらにより，現在の航空予約システムの抱える課題をカバーすると共に，より戦略的な顧客中心のシステム基盤を提供することを可能としている．

表 1 AirCore の目標と実現方法

目　　標	実現方法
■顧客サービスの向上 ■顧客ロイヤリティの向上 （課題:システムの複雑化 に対応）	従来、別なサブシステムとして位置づけられる顧客データベースを主要業務に直結した形のオペレーショナル顧客データベースとしてモジュール化し、顧客と接するあらゆる場面での顧客アプローチ機能を提供している。
■業務生産性の向上 （課題:コスト削減　に対応）	主要業務プロセスの自動化やビジネスロジックのパラメータ化によりコーディングを省力化している。また GUI を活用して業務生産性を高め、トレーニングコストを低減している。
■市場への早期投入 （課題:短期開発　に対応）	サービス指向アーキテクチャに基づく業務のコンポーネント化およびアプリケーションのレイヤ（層）の明確な分離（モジュール構造化）により業務変更要求への対応を容易にしている。さらに、プレゼンテーション・サービスをブラウザベースにすることでも業務変更要求への対応を容易にしている。また、サービスとメッセージを基本とする構造により、他システムとの連携も容易にしている。
■IT コスト削減 （課題:技術者の確保、コスト削減　に対応）	全モジュールを BEA Weblogic と J2EE（プラットフォーム・フリー）で構築し、モジュールの構造化を行うことにより、変更の容易性と技術者の確保を可能としている。

AirCore の目標を機能面から支える具体例として，

［顧客サービス，ロイヤリティの向上を目指す機能］

・顧客のチェックイン時に，お客様個々人の過去のやりとりを確認し，例えば過去にご不便をお掛けした場合など，「何々のフライトではご不便をお掛けいたしました．今回は無料でアップグレード・サービスをご利用いただきたく…」などといった個々人を尊重した対応を可能とする案内をシステムから生成させることを可能にする．

・空席待ちのお客様がおられる状態で，例えばこの便に接続する予定の到着便が遅れているなどで，数名分の空席が発生する可能性をシステムが検出した場合，処理に手間と時間の掛かる個々のお客様ごとの処理ではなく，出発時刻の直前まで待ち，一括して空席待ちのお客様とミスコネクトになったお客様を入れ替え，搭乗券の出力までを短時間で処理する機能．

［業務生産性の向上を目指す機能］

・便のキャンセル時など搭乗予定者を別の便に振り分ける処理をビジネス・ルールに従い自動的に行う機能，

・略語を使わない，判りやすい画面設計と，運用のビジネス・ルールや規程に不安を持った操作員を導くナビゲーション，

などが挙げられる．

3.2 AirCore 業務機能構成

AirCore の業務機能は，顧客中心の思想により，係員の業務に必要な機能を提供するだけでなく，顧客情報と業務を連動させて顧客の利便性を向上している．このため，その構成は図 1 のとおり，顧客情報を中心にモジュール化（コンポーネント化）されている．

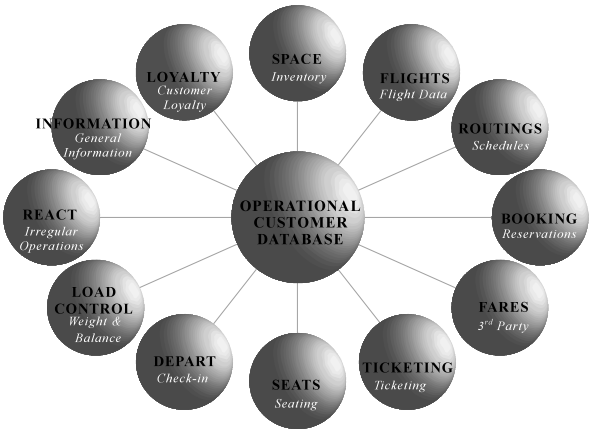


図1　AirCore モジュール構成

① OPERATIONAL CUSTOMER DATABASE
顧客情報データベースを管理する機能を持つモジュール

② SPACE
座席の在庫管理業務を行う機能を持つモジュール

③ FLIGHTS
便のフライト情報を管理する機能を持つモジュール

④ ROUTINGS
便のスケジュール情報を管理する機能を持つモジュール

⑤ BOOKING
予約業務機能を持つモジュール

⑥ FARES
運賃計算業務機能を持つモジュールで他社外部システムとの接続を想定

⑦ TICKETING
発券業務機能を持つモジュール

⑧ SEATS
座席割り当て（シートアサイン）業務機能を持つモジュール

⑨ DEPART
搭乗（チェックイン）業務機能を持つモジュール

⑩ LOAD CONTROL
重量バランス計算機能を持つモジュール

⑪ REACT
イレギュラー業務機能を持つモジュール

⑫ INFORMATION
お知らせ機能を持つモジュール

⑬ LOYALTY
フリークエント・フライヤー・プログラム（FFP）業務機能を持つモジュール

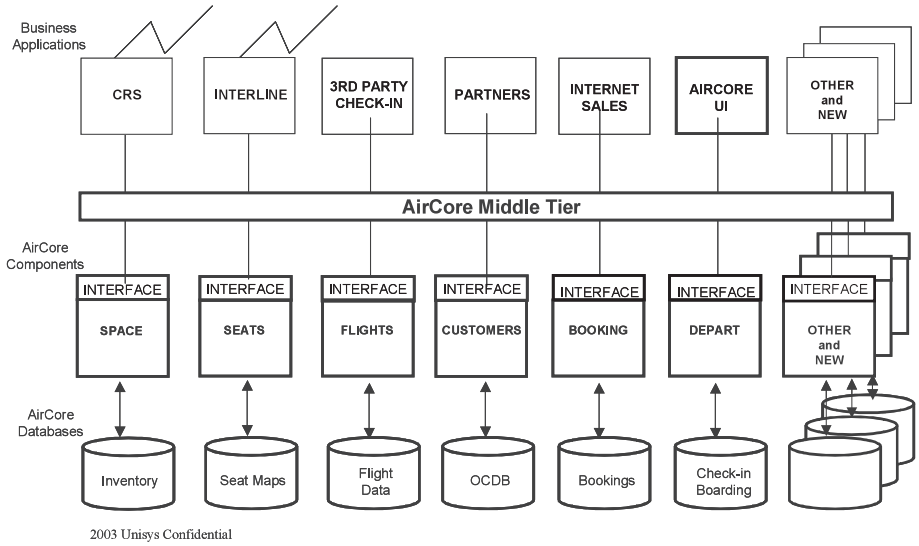


図 2　AirCore 実装モデル

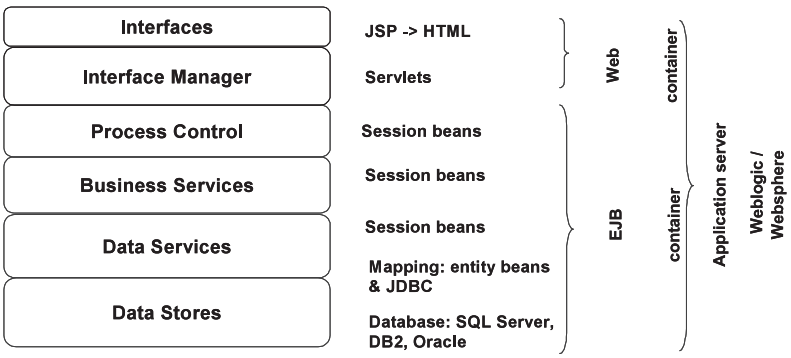


図 3　AirCore レイヤ

3.3　S/W アーキテクチャ

AirCore の各モジュールは，サービス指向アーキテクチャ（SOA）に基づき図 2 のような実装構造モデルで表現される．

各モジュールは独自のデータベースを持ち，外部のシステムだけでなく，AirCore の UI や，他のモジュールからも隠蔽され，モジュール間の連携においても，相手モジュールのデータベースを直接参照/更新するようなことはしない．図 2 に表現された Middle Tier を経由した EJB 呼出し，あるいは Web サービスなど各モジュールが公開している標準インタフェースを利用している．Middle Tier はさらに外部システムとの接続インタフェースを吸収することにより，各モジュールに呼び出し側を意識させない構造を実現している．この隠蔽と標準インタフェース利用により，SOA で目指すモジュールの機能拡張を容易にし，新機能の市場への早期投入を実現可能にする．

それぞれのモジュールは，図 3 に示すようなレイヤ（層）に分割された内部構造を持つ．

各層の役割は次のとおりである．

1) Interfaces/Interface Manager
　ユーザプレゼンテーションの層で，JSP と Servlets で構成されている．Interfaces 層は UI を実現し，下位層の Interface　Manager が標準インタフェースによるモジュール呼び出しを担う．
2) Process Control
　Stateless session beans であり，トランザクション管理をおこなう層である．この層で標準インタファースを提供している．
3) Business Services
　ビジネスロジックを処理する層で，ここにのみビジネスロジックが存在している．
4) Data Services
　オブジェクトとデータベースのマッピングを行うデータ・パーシスタンス層である．
5) Data Stores
　RDBMS（リレーショナルデータベース）層である．

このような S/W アーキテクチャ面から支援する AirCore の目標は市場への早期投入と IT コスト削減である．ビジネスロジック層への機能追加を伴わない画面のレイアウト改造は影響範囲が Interfaces/Interface Manager に限定され，開発規模を小さくし，期間短縮とコスト削減を実現する．また今後の技術革新により，JSP にとって変わる，より操作性が向上する UI の実装技術を活用する場面でも，影響は上位層だけで，業務ノウハウが実現されているビジネスロジック層への影響は最小限にした形での移行を可能にし，それまでの IT 投資と新技術への IT 投資のバランスをとり，全体的な IT コスト削減に繋げている．これはビジネスロジック層と RDBMS 層の関係でも見ることができる．データベース構造の変更だけでなく，データベース自体を RDBMS にとって変わる新技術を利用したもの，例えばオブジェクト・データベースなどに変更しても Data　Services 層で変化が吸収されるため，ビジネスロジック層への影響を最小限に抑える効果が期待でき，新技術の採用と IT コスト削減の両面を実現する手段となりうる．

4.　オープン・プラットフォーム上の基幹システム開発手法

AirCore はその開発目標を実現するため，基幹業務の構築リスクを軽減し，変更要求時の開発生産性を向上させる開発手法を取り入れている．この手法では，要求管理から実装，テストと流れる全ての開発工程にツール連携を通して一貫して管理している．この一貫した管理を通し，要求からテストに至るまで追跡検証が可能となり，変更要求時の開発生産性向上が可能となっている．この章では，その開発方式について記述する．

4.1　RUP と URUP

航空業界が抱える課題を解決すべく，次期システムの検討が開始されたのは 2000 年のことである．1970 年代にメイン・フレーム上で開発を開始した頃と大きく異なるのは，オープン・システムの出現と，業界各社がその方向性を歓迎している点であった．AirCore 開発チームとしても，アーキテクチャとして J 2 EE，開発手法として RUP[*1] の採用を決めたものの，基幹システムを稼働させるに耐えうるメイン・フレーム並みのシステム要件をオープン・システムで実現するには，長年メイン・フレームで培ってきたノウハウをアーキテクチャとそれを実装

する手段としての開発手法に組み込む必要があった．Unisys では，RUP を基本的な概念とし，これにメイン・フレームでの基幹業務系開発運用ノウハウと CMMI[*2] を取り込んだ URUP（Unisys Rational Unified Process）を Unisys 標準の開発技法として制定した．AirCore はこの手法を全面的に採用し開発されている．RUP は，要求の追跡可能性や，リスク主導の開発など優れた特徴を持つが，メトリクス管理，人的管理，テストに関しては不得手である．例えば，次のようなことである．アーキテクチャとして重要なスケーラビィリティをどのように実現するか．これに近道は無く，下手な実装が一つでもあれば簡単に，求められているシステム化要件，例えば性能面での低下などを招いてしまう．性能を実現する基本となるアーキテクチャの理解はもちろんのこと，設計段階のレビュが重要視されている．このような要件は性能評価のベンチマークを繰り返すことで実現するなど，「機能」の実装だけでは到達できない部分にも注視する必要がある．AirCore では，新しい機能がリリースされる度に実行されているベンチマークは，テーマが決まってから準備に 2 週間，実施に 1 週間，結果の評価に 3 週間かかるプロセスとなっている．これはテストデータやテスト用のトランザクション・ミックスを生成する負荷ドライバが利用できる前提での作業となり，ここまでの作業を実現するまでにも長い期間が必要とされた．このベンチマークでは新しい機能の性能評価だけでなく，新しいハードウェアや新しいオープン・システム・ソフトウェア（アプリケーションサーバやデータベース，各種ドライバなど）の評価も兼ねている．すばらしいアーキテクチャやそれを理解している設計者，ベンチマークによる評価を行うのは，オープン・システムを利用するから必要になったわけではなく，メイン・フレームでも技術は異なるが実演してきた内容で考え方も変わっていないからである．利用するソフトが自社のプロプラエタリの組み合わせから，第三者のオープン・システムのものに変更されただけで，それらを利用するノウハウが重要なのは同じである．

このように AirCore の特性に対応するため，URUP を UCM[*3] と一体化して，プロジェクトマネジメントと一体的なツールを用意し，開発手順に構成管理とプロジェクト管理が組み込まれているのが，この開発手法の特徴である．

4.2 URUP の狙い

URUP は RUP の良いところを継承している．

4.2.1 リスク駆動開発

URUP は大規模開発におけるリスクの軽減を狙い，「イテラティブ開発」（図 4）を推奨している．従来のウォータフォールモデルでは，開発の最終段階にならないと，実装モデルの検証ができず，期待通りの機能や性能，操作性などの要件が満たされるか不明であった．イテラティブ型は一つの開発を数週間から数ヶ月規模の複数のイテレーションに分割し，一つ一つのイテレーションで評価可能な成果物を作り検証を繰り返すことで，早い段階でのリスク軽減を主眼においている．そのため，スパイラル型のように作りながらシステムを精査するのではなく，リスクの高いテーマから作業していくことをその方針としている．

4.2.2 イテレーションとフェーズ

イテレーションは「反復」あるいは「繰返し」とも訳され，図 5 で示す開発では 10 回のイ

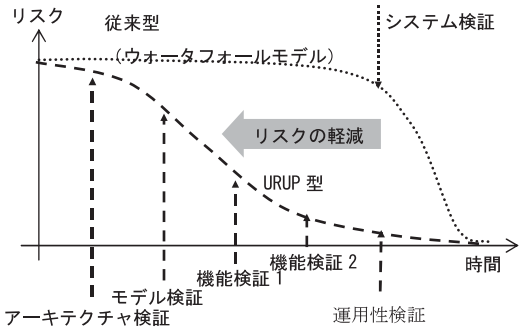


図 4　開発型とリスク

テレーション（下段の箱）で行われたことを表している．一つの開発を複数のミニ開発の集合に分割しているとも言えるが開発の経過とともに目標が変化する 5 つのフェーズ（図 5 の上段の箱）に分けられている．

・Inception（方向づけ）
・Elaboration（推敲）
・Construction（作成）
・Transition（移行）
・Production（運用）

と名づけられたフェーズでは開発の各作業項目（ここでは縦軸の Discipline,「規律」あるいは「修行」）の負荷の山（高い＝高負荷）が開発の推移と同時に作業の中心を移動させている．すなわち初期の「方向づけ」のフェーズでは，Business Modeling（ビジネス設計）や Requirements（要件定義）などの「何をするか」の山が高く，Analysis & Design（分析と設計）や Implementation（実装）などの「どう実装するか」が低くなっている．しかし，フェーズが進むにつれて段々と実装に移っている．一回のイテレーションとしては，山が低くとも，常に両方の要素が含まれているのが URUP の考えになる．Test は各イテレーションの最後に行われる機能テストを指しており，Deployment は本番移行に向けた作業になる．Configuration& Change Mgmt（変更管理），Project Management（プロジェクト管理），そして Environment（開発環境）はプロジェクトを支援する作業項目になる．最後の Operation ＆ Support は運用と支援を指し，本番を迎えたシステムもソフトウェアのライフサイクルに含めている．次のフェーズに移るには，必ずプロジェクトで定義したマイルストーンに達したことを確認してから先に進むことが重要視されていて，必要であればイテレーションを再度実施し，マイルストーンをクリアすることがプロジェクト推進上期待されている．

4.2.3　フェーズと資源配分

これをフェーズ別の負荷や期間の割合としてみると従来型の開発リソースの配分と類似する（図 6）．違いは個々のフェーズ内で従来型の開発がイテレーションの名称で行われ，全体として機能している点にある．

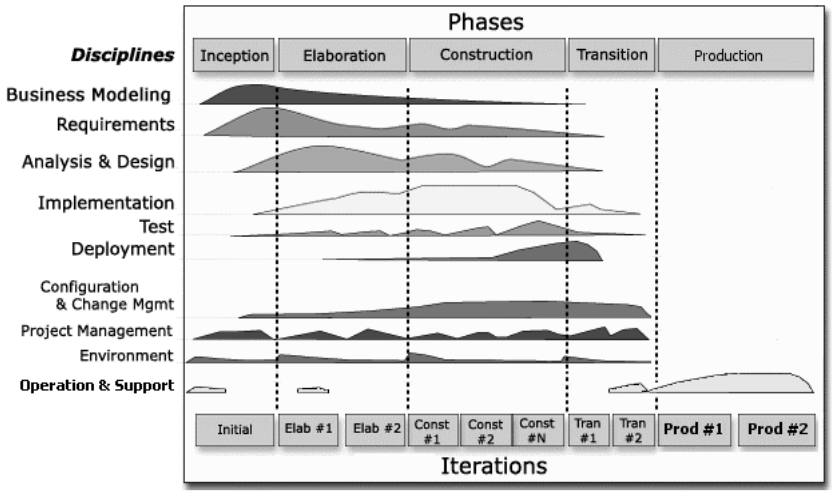


図 5　URUP のフェーズ

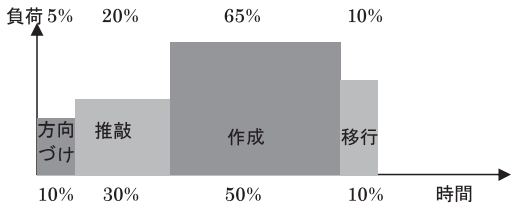


図 6　URUP と資源配分

4.2.4　規律とフェーズ，イテレーション

イテレーションとフェーズの図（図 5）に表されている「Discipline＝規律」は，

① ビジネスモデリングと要件定義
② 分析・設計と実装
③ テストと導入
④ 変更管理，プロジェクト管理と開発環境
⑤ 運用と支援

に分類することができる．①は業務寄りの要件定義に位置づけられ，②が従来の「システム開発」，そして③が本番を迎えるためのシステムテストと本番環境へのデータ変換やトレーニングを含む導入作業に該当する．特に③の「テスト」はシステムテストだけでなく，各イテレーションの最後に実施される結合テストあるいは総合テストも示しているため，導入直前だけでなく，各イテレーションにおいても作業負荷の山が現れている．これはイテレーション毎に「動作する機能」を実装させることを目的とし，成果物を常に「検証」する URUP の基本的な考えからきている．④は開発期間中常に背景で行われている作業だが，プロジェクト管理の山が最初の方向づけのフェーズにあるのは，ここでプロジェクトの詳細計画が作成されているからで，従来のシステム化の企画や検討の段階も包含していることを意味している．リスク軽減の観点から，ここではシステムと想定業務の剥離のリスクに焦点をあてている．

推敲のフェーズでは管理面より，分析と設計の山が大きくなっており，どのようなシステムアーキテクチャを利用するか，どのような技術基盤の上にシステムを構築するかなどといった非機能要件を満たすための考察も，本格的な開発となる作成フェーズに入る前に行われている．ここでは主要な技術リスクを低減させる．変更管理の山が作成のフェーズに入ると高くなるのは，実装される機能の増大に従い管理すべき成果物が増えるだけでなく，イテレーションにおける検証の繰返しにより，当初想定されたイメージが実装されることによって表面化する課題を次のイテレーションで対応させるための変更管理が増えていくことを意味している．URUPでは要件の変更を早い段階で表面化させ，対応することを変更管理で実践する．

4.3　RUP プロセスのカスタマイズ

URUP は基幹システム開発を目指し，RUP プロセスをカスタマイズしている．これはフェーズやプロセスの追加，各種ガイドラインの整備だけに留まらず，ツールがより密に連携に情報の継承が実現されるようにするところまで含まれる．

4.3.1　URUP とワークフロー，ガイドライン

URUP の元となった RUP はバイキング形式の料理とも対比されるが，ここにはベストプラクティスと評価されたあらゆる種類の技法や手順，成果物の定義が存在しており，目的とするシステムを構築するには有益なものを選別し，余分な作業負担とさせない選択眼が求められる．URUP はその中から企業の基幹業務の構築を視野に仕立てあげたもので，URUP が目指す方向に合わせて RUP の考えを取捨選択し，ガイドラインや手順に Unisys の過去の経験を盛り込んでいる．たとえば，URUP のフェーズに「運用」，そして Disciplines に「運用・保守」を追加した部分（図 7）も URUP 独自の仕立て部分である．

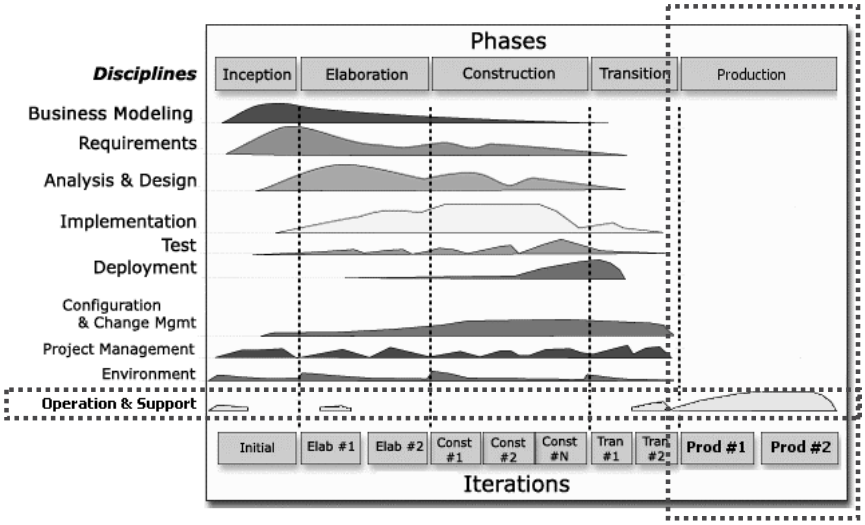


図 7　URUP のフェーズ

URUP では RUP 同様に，役割別の作業が定義されており，個々の作業ごとに入力となる成果物（前の作業の成果物など），作業手順，その役割の人が責任を持つ出力成果物，そしてそ

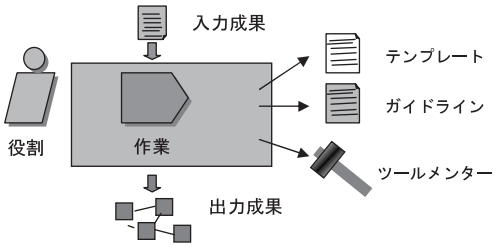


図 8　URUP の構成要素

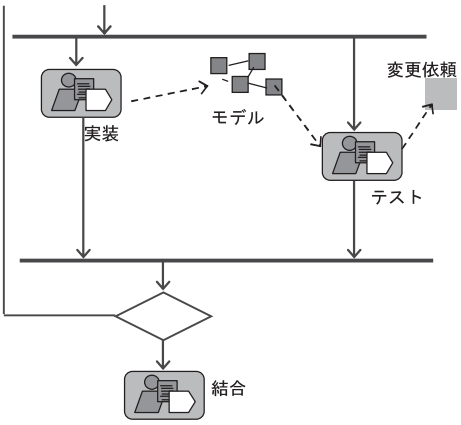


図 9　URUP のワークフロー

の作業を支援するために関係するテンプレート，ガイドライン，そしてツールの具体的な利用方法を伝授するツールメンターが参照できる（図 8）．これらについても，URUP 独自の物が用意されている．

これらの作業はワークフローとして関係が定義されており，各フェーズの各作業間の流れとして利用することになる（図 9）．このフローは個々の作業や成果物がクリック可能となった Web サイトとして提供されており，プロジェクト参加者はいつでもこのサイトへアクセスし，必要とする情報，ガイドライン，手順，考え方の説明文や成果物のテンプレートなどを参照することができる．

この共通の開発手法以外に，AirCore 独自の資料としては開発・保守に必要な一般的なガイドライン（コーディング規約，ネーミングルール，DB アクセス層作成ガイドなど）が一式 AirCore プロジェクト用に用意されている．これらはソースレベルでカスタマイズする場合には必要な情報となるが，これは Knowledge Base として別サイトで管理されている．

ここには先に述べた通り CMMI に準拠させるための考えも取り入れられている．

4.3.2　AirCore の成果物

URUP を利用して作られた AirCore の主要な成果物は次のとおりである．

1）　ビジョン・ドキュメント

これは AirCore の各モジュール（SEAT,REACT など）ごとに存在し，ここには主要機能の全体概要が記載されている．モジュールに含まれる個々の機能説明やサブ機能に細分化された機能説明の下にユースケースが存在している．ビジョン・ドキュメントは URUP

のビジネスモデリング作業で作成され，一つのビジョン・ドキュメントは従来型システム開発における一つのサブシステムと対比することができる．そこにはそのモジュールが対象とするビジネス面を定義するプロジェクトの企画書や構想書の内容を含む．あくまでビジネス面からのアプローチとなるため，システム的な概念は含まれない．AirCore では Word のファイルとして作成されている．

2) ユースケース・ドキュメント

ビジョン・ドキュメントをベースに機能の詳細化を行うとともに，業務の流れや手順を定義しユースケース・ドキュメントとして要件定義段階で作成する．ここでは関係するアクター（ステークホルダ）や処理のオブジェクトが抽出されるが，ビジネスのモデル化の一環として行われ，ビジネス・ユースケース・モデルとして，システム的な要素が入らない概念的な定義資料として作られる．AirCore では画面遷移や操作手順を図式化するためストーリボードをこの段階で作成する場合もあるが，補助的な資料として位置づけられ，イテラティブ開発で随時作成される実画面が成果物となる．ビジョン・ドキュメントと個々のユースケース・ドキュメントがシステムの開発を承認するステークホルダへの説明文書となる．ビジョン・ドキュメントをサブシステム相当と位置づけるなら，ユースケースがプログラム相当と言え，以降はユースケースが管理単位になる．

3) UML モデル（ユースケース図，シーケンス図）

ユースケース・ドキュメントは分析段階でユースケース・モデルとなり，設計段階で実際のクラス図へと繋がる．ここで実装されるシステム的な設計に着手するが，オブジェクト指向の設計を基本としているため，従来の外部設計書とは大幅に異なる．AirCore のカスタマイズや機能追加を行うには，これら UML モデルを理解する知識が前提となる．カスタマイズや保守において，該当するユースケースからモデルを絞り込むには AirCore のコンポーネントが持つ機能の知識が求められるが，クラスの特定ができれば後は UML のモデルから処理の流れや関係を調べ，UML モデルツールを用いてドリルダウンしていくことができ，変更作業における影響度分析も含め，ソースコードベースからモデルを用いての作業になっている．処理内容の表現にはシーケンス図が用いられる．これも分析で作られたモデルを設計で詳細化するステップが取られる．これらが従来開発での詳細設計書に該当するが，違いはモデル＝ドキュメントとなり，保守用のドキュメントを別途用意する必要が無いことは保守性の面からも評価される点である．

4) ソースコード

実装ステップで作成されるが，基本的なスケルトンやインタフェースはモデルからツールで自動生成され，モデルと実装の整合性を保つことができる．一般的ではないが，ソースコードを先に修正した場合には，リバースエンジニアツールでモデルに反映させることも可能である．AirCore が利用しているアーキテクチャでは，オブジェクトを DB に保存するための OR マッピングはデータ・パーシスタンス層で行われており，ビジネスロジックを実装する層では DB の構造に関する知識は必要なくデータの保存方法からは完全に独立している．

4.3.3 URUP と UCM 連携

UML 設計ツール（Rose）から見たモデルは構成管理ツールと連携し，上位のビジネス・ユ

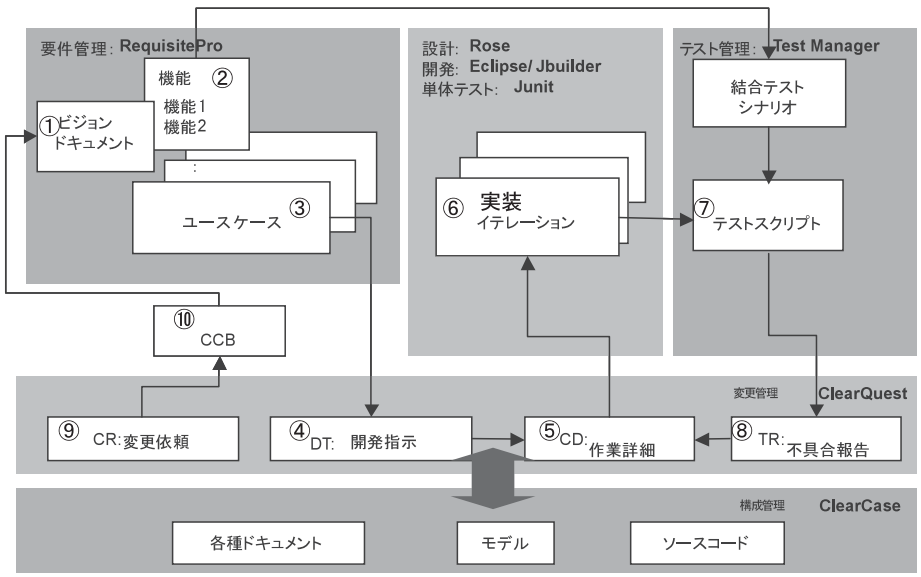


図 10　UCM で利用されるツールと開発ツールの関係

ースケース・モデルと紐付いて管理され，ツールと一体化した利用が前提となっている．開発全体を見た UCM で利用されるツール（Rational　Tool　Set）と開発ツールの関係は図 10 の通りである．

URUP で定義された UCM では変更管理で管理される各アクティビティを通じて処理の流れが制御されている．開発で利用される DT アクティビティの処理フローを URUP の作業ステップに対応させた処理の流れは次の通りである．

1）　業務定義（Business Analysis）

BA（Business　Analyst）が要件を固め，ユースケースとして業務の流れと内容を定義し，ビジョン・ドキュメントとしてシステム化の全体像を定義する①．ビジョン・ドキュメントモジュールは RequisitePro に取り込まれ，機能単位に管理が分割②されるとともに，個々の機能はハイレベルのユースケースとして表される③．この処理を行うために，ビジョン・ドキュメントは Word のテンプレートとして用意されたものを利用する．このテンプレートにはオブジェクトが埋め込まれており，RequisitePro が解釈できる仕組みとなっている．実際に，RequisitePro から Word のドキュメントの関連箇所へのリンクがされており，目次や HTML のリンクのようにワンクリックで移動することができる．ここでは，「ビジネス」の言葉で表現するが次の機能定義の段階で必要となる「アクター」も明示し，外部（ユーザやシステム）と関係者（ステークホルダ）の役割や利害関係を整理するのも重要となる．

ビジョン・ドキュメントが作成され，RequisitePro に取り込まれると，そこからプロジェクトの進捗管理が開始される．ビジョン・ドキュメント自体は構成管理ツールである ClearCase で管理されるが，そこへの新規登録や変更のためのチェックアウトは ClearQuest からの作業指示が必要となる．一つ一つのユースケースや機能には DT（Development Task）が PM により作成され④，作業担当に割り当てられる．この作業指示は ClearQuest により管理される．

2）　機能定義（Requirements）

SA（System　Analyst）がビジョン・ドキュメントのユースケースをモデル化する．ストーリボードで，ユーザが操作をイメージできるようにする場合もあるが，これはあくまで説明用に利用される補足資料であり，成果物としては管理されない．画面はあくまで，実画面（JSP ファイル）として管理される．シーケンス図は文字で表現された各処理の流れをモデル化するのに利用される．ここでは，言葉で表現されている業務からオブジェクト指向のモデル化技法を用いてオブジェクトを抽出するのが主な作業である．基本的にここが開発の依頼者との機能確認を行う段階である．AirCore の開発ではビジネス面の要件や機能を定義している担当者は AirCore の開発センタの人間であり，システム化技術の背景にも明るいため，次の詳細モデル化の分析部分も実施している．通常の開発のようなエンドユーザ部門を交えた体制の場合は開発における役割の明確な区分が必要になると思われる．

3）　詳細モデル化（Analysis & Design）

ここは従来のシステム開発での外部，論理設計に相当する．実装のステップになる⑥では Rose を用いて分析し，結果をユースケース単位に詳細モデル化する．作業の単位は全て CD（Change Detail）⑤として管理される．この CD が登録されていないと，変更のために既存のモデルをチェックアウトしたり，新規にモデルを登録することができないように，ツール間の連携がされている．

ユースケースがプログラムに対応し，管理もこの単位で行われる．ここではオブジェクトの詳細化が行われ，属性の定義，再利用，別クラスの変更依頼が分析される．オブジェクトがいかにきれいに設計されるかはここを担当する人のモデル化技術力に依存するが，それ以上に，全体システムのアーキテクトを定義したシステムアーキテクトのレビュ能力にも大きく依存する．モデルの成果も構成管理のリポジトリに保存され，保守対象となる．このモデルは java ソースを直した場合でもリバースエンジニアによってモデルに反映される．変更が発生した場合に，修正箇所の特定もこのモデルを利用するため，従来のようにソースを読んで理解するケースは基本的に無くなる．オブジェクト指向の設計と，実装の鍛練は必須である．モデルと実装の差，コーディングの役割，単体テストの価値など，「モデリング＝実装」とならない現状での課題は，オブジェクト指向開発に慣れていない技術者の場合，実際のミニ開発を通じて肌で感じる必要がある．

モデルからソースのスケルトンを自動生成し，クラスの継承による共通処理の確実な埋め込み技術を用いると，ログ，エラー処理，統計情報，セキュリティチェックなど，コーディング規約や開発標準でプログラマ個々の規範として維持してきた部分が不要になってくる．とは言え，オブジェクト指向言語である java を手続き型の記述でコーディングすることも可能で，どのように利用するかは基本的なシステムアーキテクチャとして定める必要がある．すなわち，java＋EJB＝オブジェクト指向ではなく，そこにオブジェクト指向の“規範”が存在していなくてはならない．この“規範”は AirCore 内に存在するが，法律のように明文化されたものではなく，文化のようにそこに生活して初めて感じることができるモノであり，そのなかで自由に活躍できるようになるには時間が掛ることを認識しておく必要がある．

変更依頼である CR（Change Request）はまず影響度分析が実施され，その成果と合わ

せてレビュ機関である CCB（Change Control Board）にかけられる．ここで関係者のレビュを受け，変更が妥当であることの検証が行われる．これは，システムを長期に渡り維持管理するため，安易な妥協を行わないための監視機構として機能するだけでなく，開発の余分な手間を省くため，既存オブジェクトをリユース（再利用）させることにも留意している．

CR が承認されると，CD が作成され，その後，それをどのイテレーションで開発するかはプロジェクト管理における優先順位に従って決定される．まだ作業を具体的な計画に取り込まないものは HOLD（保留）状態に置かれて管理される．現在 AirCore においても，HOLD 状態の機能が存在し，これらは必要に応じて将来の拡張機能として実装される可能性がある．

4） コーディング・単体テスト（Implementation）

実装ステップでは分析・設計ステップで定義されたクラスとメソッドの実装を行い，単体テストまで実行する．各実装担当者は個人の開発環境上に担当モジュール全てのソースのコピーを保持し，単体テストとデバグに利用する．このソースを常に最新に保つのは構成管理ツールの機能だが，同期を取るタイミングは開発者が自分の都合に合わせ実施することができる．

単体テストのため，開発者のワークステーションにはアプリケーションサーバ，AirCore の全ての EAR ファイル，MQ クライアントなど DB を除く全ての環境が搭載されており，相互に影響することなくテストが実施できる．コーディングやデバグは JBuilder を現在利用しているが，特定の Java IDE に依存していない．一部の開発者は Eclipse も利用しており，今後 Eclipse の利用も増えると予想される．

AirCore は常に業界の最新技術を取り込むことを前提としているため，業界標準に準拠することを意識している．開発ツールが変更可能なのも，特定のツールに依存する仕組みを極力避け，常に最新技術の恩恵に与れるように配慮する姿勢が現れている．

5） 結合テスト（Test）

単体テストを終え，完成したソースを ClearCase に保存すると同時に，CD に対し完了の設定を行う．一つのイテレーションで予定している DT に紐ついている全ての CD が終了したら，その DT が結合テスト⑦の対象となる．ClearQuest で管理されているステータス情報は通常の RDB であり，利用者がクエリを発行し，必要とするレポートを作ることができ，そのイテレーションで開発中の各 DT の進捗は完了した CD の数で管理される．作成したクエリを適時実行し，状況を確認していくのは PM の役割となる．テストケースの管理に Test Manager を利用している．

結合テストは実装チームが利用している開発環境とは別に管理されたストリーム（構成管理の単位）で行われる．テスト用ストリームへの単体テストを完了したソースのマージは構成管理チームが実施する．ここで実装担当がバラバラに作業していた環境が一つにマージされ，結合テスト用として全てコンパイルされビルドされる．このマージは自動的に行われるが，不整合やエラーが検出されると実装担当者に通知され，担当者が解消する．AirCore の場合，このマージ作業は変更の範囲により，通常半日から 1 日程度時間が掛かる．結合テストはモジュール単位に行われ，リリース可能な状態になれば，AirCore 全体の結合テストストリームにリリースされ，他のモジュールのチームが利用可能となる（図 11）．

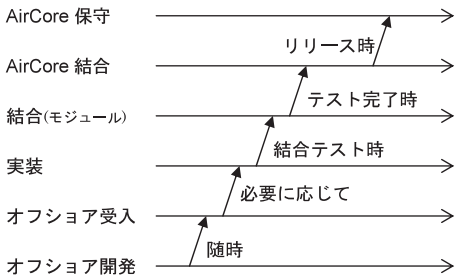


図 11　ストリームをマージする流れ

結合テストはテストスクリプトによって実施されるが，このスクリプトはユースケースが定義されると同時に，実装チームとは別にテストチームが作成するテストシナリオをベースに作成される．AirCore では各モジュールが EJB 呼び出しで利用されるサービスと同時に UI も開発されるため，テストはまず，EJB 呼び出しをテストスクリプトによって実施し，機能の検証が済んでから UI を利用したテストを実施する．スクリプト化の範囲はスクリプトの作成，保守の負荷と利用価値で決められ，全てをスクリプト化することはしていない．テストで検出された不具合は TR（Trouble　Report）として ClearQuest に登録される．実装チームにより原因が解明されると CD として再度登録され，修正，単体テストを終えてから再度結合テストが実施される．結合テストは，実装チームとは異なる構成管理のストリームとして管理されている．

このように，AirCore が利用した URUP の作成フェーズで開発が完了するまでイテレーションを繰り返し実施されるが，作成フェーズをスムーズに進めるためには，推敲フェーズで性能や品質などのシステム化要件を満たせるシステムアーキテクチャが検証も含め確定していることが一番重要となっている．

いかに採用したアーキテクチャが優れていて，最新 IT 技術を採用してもそれだけで最良のシステムが保証されたわけではない．それを実際に作り上げるには有効な開発手法が必要とされている．AirCore も，採用したアーキテクチャの良さとそれを実際のシステムとして作りあげた開発手法が一緒になって初めて実現したと言えるだろう．

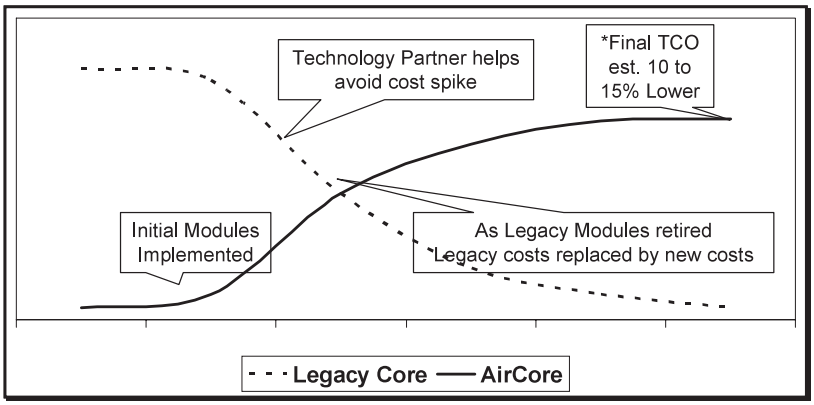
5.　期待される効果

USAS から AirCore へ Unisys のエアライン・パッケージが世代交代することによって，Unisys が目標として掲げた，

・顧客サービス/ロイヤリティの向上
・コスト削減（業務生産性及び開発生産性の向上）
・短期開発による市場への早期投入

が図れることに加えて，サービス指向アーキテクチャに基づいた AirCore を活用することによる効果も期待できる．すなわち，企業内外の他システム（例えば，企業外として旅行会社のシステムや，顧客企業が持つ出張サポートシステム等）から Web サービスによる連携を行うことで，作りこみによる硬直化したシステム連携ではなく，企業間システムの柔軟な構造化を実現し，ビジネスに変化対応力をもたらすことが可能となる．

さらに，開発手法として URUP を採用することにより開発リスクが軽減され，提供される



*Note: Gartner study of Distributed Object Computing environments estimate 10 to 15% lower Total Cost of Ownership than Legacy. This would need to be analyzed with the individual airline to arrive at estimated actual for that airline.

図 12　AirCore とレガシーシステムの TCO 比較

テンプレートやガイドラインを使用し規律を守ることで品質維持を図ることが可能である．

また，Gartner はレガシーシステムに比べ最終的に TCO（Total Cost of Ownership）が 10〜15% 削減できると想定している（図 12）．この図は，レガシーシステムが徐々に AirCore に置き換わることにより，レガシーシステムのコストが減少し，かわりに AirCore のコストが増加していくが，最終的には当初のレガシーシステムの TCO に比べ 10〜15% TCO が低くなると想定されることを表している．

6. お わ り に

現在 AirCore は，ルフトハンザ航空にて GDS からの空席照会機能が稼働しており，平均 120 件/秒の性能を出している．その他のモジュールについては，TICKETING を除きシステムテストの段階に入っており，TICKETING を含めて 2006 年 6 月に稼働の予定である．したがって，AirCore が大規模ミッションクリティカル・システムとして真の評価を得るのは，2006 年度以降となる．また，今まで述べてきた将来への変更の容易性が，本当に現時点で見通している範囲なのか全く異なる変更要求が出現するかはこれからの AirCore 自身の評価と共に，実際の運用実績を待つ必要がある．レガシーシステムからの移行リスクを考慮してどのように移行するかも重大な課題となるであろう．イテレーションを用いた開発手法の適用も，新たに参加する開発要員の育成を含めた開発環境，体制の整備を如何に行うかも大きなポイントである．これら実際の適用を想定した利用顧客それぞれの状況の把握なくして成功は望めない．さらに，そこでの開発生産性は，習熟期間を考慮した上で想定する必要がると思われ，如何に開発計画を引く事ができるかも問われる技術である．この辺りの適用技術の検討は，本稿の範囲が AirCore 紹介であることと，日本における具体的な導入案件がまだ存在しないことから，今後の課題として留めておくこととしたい．しかし，イテレーションによる評価と改善を繰り返すことが，今後の S/W アーキテクチャや開発手法として広く確立されることを想像することは難しくない．

最後に，本稿執筆にあたりご協力いただいた Unisys および日本ユニシス・ソリューション（株）の関係各位に深謝するものである．

＊1　ラショナル統一プロセス
＊2　Capability Maturity Model Integration　能力成熟度モデル統合
＊3　ラショナル社の統一変更管理

参考文献　［1］ IBM 1999 World Airlines benchmark report by IDL
［2］「RUP 実践者を成功に導く　ラショナル統一プロセス〈RUP〉ガイドブック」
パー・クロール，フィリップ・クルーシュテン　星雲社　2004/11
原書：The Rational Unified Process Made Easy : A Practitioner's Guide to the RUP by Per Kroll and Philippe Kruchten Addison-Wesley Professional

執筆者紹介　佐　藤　　覚（Satoru Sato）
1982 年日本ユニシス(株)入社．CAD/CAM システム開発，1986 年より航空予約システム，発券システム，収益管理システムなどの開発を担当．その後，旅行関連システム開発業務を経て，現在，社公システム一部エアラインシステム四室長.

小山田　和 人（Kazuhito Oyamada）
1970 年日本ユニシス(株)入社．1976 年より航空予約システム，収益管理システムなどの開発を担当．その後，航空業，旅行業などのシステム提案業務を経て，現在，社公システム一部にて，次世代航空予約システムの検討に従事.

平　松　敦　郎（Atsuro Hiramatsu）
1977 年日本ユニシス(株)入社．サポートセンタにて ASM,FORTRAN,COBOL,RPGII などで作成されていた汎用機上システムのコンバージョンに従事．UNIX ワークステーション上の CAD/CAM システムに携わり，C,XWindows,PC-UNIX で開発．SYSTEMν で CORBA を経験以降 PSA,SRM,SCM,ERP などの新技術の技術評価を経て，現 AirCore センタ長.