松本　吉弘：　サービス・サイエンスで変わるソフトウェアファクトリ（ Perspectives of Software Factory）；日経 BP 社 ITpro 2008/4/14-19 のために作成した原稿

サービス・サイエンスで変わるソフトウェアファクトリ

松本　吉弘　（工学博士）
京都高度技術研究所

第 1 章：　新しいソフトウェアファクトリが目指すところ

歴史

1968 年、ドイツ・ガルミッシュにおいて、第 1 回ソフトウェアエンジニアリング・ワークショップが開かれ、「ソフトウェアエンジニアリング」という語が初めて鋳造されたとされている。そのとき、M.D. McIlroy[1]は、再利用によってソフトウェア生産性、品質を向上するための組織を、ソフトウェアファクトリとよぶことを提唱した [McIloy69]。

わが国では、1960 年代末から 1970 年代初めにかけて、表 1 に示す各社が、それぞれのソフトウェア事業所をソフトウェアファクトリ（またはソフトウェア工場）と位置づけ、独自のマネジメント体制を確立した。

表 1　日本の旧世代ソフトウェアファクトリ

設立年	企業名	施設	製品	従業員数
1969	日立製作所	日立ソフトウェア工場	BS	5000
1976	日本電気･ソフトウェア戦略プロジェクト	府中事業場	BS	2500
		三田事業場	RT	2500
		三田事業場	App	1500
		我孫子事業場	Tel	1500
		玉川事業場	Tel	1500
1977	東芝	府中ソフトウェア工場	RT	2300
1979	富士通	システム本部	App	4000
		蒲田ソフトウェアファクトリ	App	上記中1500
1983		沼津ソフトウェアファクトリ	BS	3000
1985	日立製作所	情報システム工場	App	6000
				内訳:SE:4000
				内訳:PG:2000

注：　BS: basic software; RT: real-time systems; App: application software; Tel: telecommunication software

70 年代の米国では、Software Development Corporation (SDC)が、全社をソフトウェアファクトリ化する試みをし、IBMは、サンノゼに、Santa Teresa Laboratoryを発足させた。また、AT&Tは、Programmer's Workbench System (PWB)を開発し、全事業所に適用しよ

[1] http://www.cs.dartmouth.edu/~doug/components.txt

うとした。80 年代になって、V. Basiliらは、「経験」の再利用によってソフトウェアの生産性、品質を向上するための手法Experiment Factory (EF)を開発し、90 年代に入って、有力企業の事業所に適用しようとした。これら 20 世紀後半のソフトウェアファクトリは、旧世代ソフトウェアファクトリと呼ばれている[2]。

パラダイムの変化

90 年代に入って、ダウンサイジング、開放型共通基盤のグローバル化などに伴い、旧世代ソフトウェアファクトリは、それぞれさまざまな形で大きな変貌を遂げてきた[3]。

新しいソフトウェアファクトリが追求している第 1 の課題は、ソフトウェア開発・保守における新しいパラダイムへの対応である。新しいパラダイムを、旧世代のそれと比較して、表 2 に示す。

表 2　ソフトウェア開発に対するパラダイム変化

	これまでの考え方	潮流となるこれからの考え方
設計プロセス	計画駆動、線形、連接	繰り返し、発見的、やってみて拙ければやり直す
目標	最適化	適応性、融通性、即応性、可変性
問題解決手段	論理指向、形式性、証明可能、	意味指向、レビュー、運用経験による学習、リファクタリング、リエンジニアリング、進化型改善
周囲の状況	安定、予測可能	変動、不可解、予測不能
適応方法	一発解決を目標	繰り返し適応を前提
基本特性	すべてコントロールできるはず、実装以前に設計がなければならないという思想	協業、コミュニケーション重視、設計と実装を区別しない、ファシリテーション重視
根底思想	技術優先、普遍化可能	不確実的、主観的
頼れるもの	論理的、科学的アプローチ	実践経験による学習、および適応のためのガイドライン

パラダイムとは、なんらかの目的を追求するために体系化される、一貫した思想・概念の流れを意味する。これまでのソフトウェア開発・保守パラダイムは、たとえば構造化設計のように、定義された要求のセマンティクスに対して真になり得るような論理構造を組立て、これによって安定し、安全で欠陥を予測できるシステムを構築しようとするものであった。しかし、新しいパラダイムは、いろいろな形で「複雑・多様化し、かつ不連続に

[2] http://www5d.biglobe.ne.jp/~y-h-m/IntroductionToSF.pdf
[3] 日経 SYSTEMS、2006-9, pp.61-68

変化するビジネスダイナミクス、またはシステムダイナミクスへの即時適応」を求めている。「モデル」がなぜ、使われるか？複雑・多様化した要求のもつ意味の属性と静的構成を可視化するためである。しかし、モデルはあくまでも意味の表現であり、それを「プログラム」と呼ばれる論理へ変換するためには、制約克服や動的振る舞いの組込みを必要とする。このための方法論のひとつとして、デザインパターン（およびフレームワーク）を利用するマイクロソフト社のSoftware Factories[4]と称する手法があり、実用化へ向けての努力が進められている。しかし、多くの現場では、従来どおりの計画駆動型開発を進めながら、その途上で、要求ダイナミズムに即応するための修正を適時に繰り返す、という進化型開発（revolutionary development)、または漸次拡張型開発（incremental development）が実践されている[5]。

ソフトウェアファクトリにとっての第２の課題は、「価格が人月ベースで決まるという商習慣のもとでは、ソフトウェア開発・保守の効率化・自動化・工業化は、ソフトウェア企業の売上げ低下、利益減少を招く」という現実の克服である。これに対する解決を見出さねば、ソフトウェアファクトリは、いつまで経っても、成功ビジネスを手にすることはできない。

サービスへ向かうエンタプライズ･ソリューション

まず第１の課題を取り上げる。昔からソフトウェア開発・保守に、銀の弾丸（silver bullets）はないとされてきた。しかし、銀メダルくらいは目指したい。新しいパラダイムに向けての挑戦には、大きな目玉が２つある。すなわち、（１）消費者参加型開発、またはプロシューマ参加型開発（producing consumerープロシューマ)、および（２）サービス指向開発である。ここでいうプロシューマとは、エンタプライズ（システム、またはプラント）のなかで、ソフトウェアを利用して業務を遂行する組織のメンバーを指し、ソフトウェア知識・スキルをもっている必要はない。プロシューマが直接行う開発を、即席開発（opportunistic development[6]）とよぶ。ユーザ開発と呼んでもよいが、ユーザという語は定義が難しいので、本稿では即席開発（これは筆者の作った訳語）という語を用いる。新しいソフトウェアファクトリは、即席開発と、専門家が行うシステマティックなサービス指向開発（systematic service oriented development）を組み合わせることによって、プロダクトを開発・保守できるようにする。

即席開発の分かりやすい例を示そう。太郎君は、ある企業の海外プラント担当のビジネスマンである。太郎君は、3 ヶ月に１回は、外国出張をしなければならない。太郎君の希望は、出張のたびに必要となる、煩雑な出張申請、旅費申請、外貨購入、航空券予約、ホテル予約、顧客との会合設定、帰国後の業務報告、旅費精算などを、できるだけ自動化し、

---

[4] Greenfield, J., et al., Software Factories, Wiley (2004)
[5] Forsberg, K., et al., Visualizing Project Management, 3rd Ed., Wiley (2005)
[6] http://msdn2.microsoft.com/en-us/architecture/bb906058.aspx

さらに、出張中に起きる異常（たとえば、フライトのキャンセル）をリアルタイム・イベントとして自動的に入手して、フライト変更、ホテル変更、会合変更を自動的に即時実行できるようにしたい、というのである。ソフトウェアファクトリの役割は、太郎君が自分で行う、ごく簡単でわかり易い入力（これが即席開発におけるプログラム：表や図で表現する）を実行することによって、太郎君の希望を満たすようなアプリケーションが自動的に生成され、作動し、太郎君の要求が達成される。裏舞台であるアプリケーション・ソフトウェアでは、外部調達可能なサービスを最大限利用する。

「サービス」という語は、消費と生産を一体化させるガソリンステーション・セルフサービスから、web service, service-oriented architecture (SOA), SaaS (Software as a Service), S+S[7]に至るまで、幅広く使われている。ここでは、「サービスとは、人の知的活動による実世界に対する価値創造（改善を含む）を支援するために、第三者によって提供される支援である。」と定義する。

受注建築・施工では、個人建築主の要求は、時代や環境とともに多様、かつ奔放に変化する。施工者は、建築主の要求をすべて受け入れなければならないが、工法、建築材料、建築用資材、インテリア製品、開口部材、照明・音響機器などは、その都度、新規開発するのではなく、外部調達可能な商品、とくにロングテール・プロダクト（数は出ないが、特定の専門分野では傑出した商品）などを利用して、適切に要求に応えている。このようなプロダクトの価格は、人月ベースで決まるものではなく、プロダクトのもつ市場価値によって定まる。この事業の基礎にある概念のひとつが、パレート（V. Pareto)の 80-20 法則[8]の適用である。進化（evolution）を前提とするソフトウェア開発・保守における「サービス指向」を支える思想は、80-20 法則であるとされている[9]。

- 外部調達可能な構成部品（オープン・ソースやサービス）で、80-20 法則を適用可能なものをできるだけ選択し、アプリケーションのなかで自由に差し替えできるようにする。
- 80-20 法則を活用し、顧客の関心がきわめて高い部分（20%）を識別し、そこではとくに高い満足度を得るような開発・保守を実施し、それ以外の部分については必要以上に微細な要求を追いかけない。
- エンタプライズやアプリケーション領域の共通特性を、80-20 法則の考えを参考にして識別し、共通フレームワークを開発し、再利用する。
- プラットフォーム（OS、言語、通信基盤、ハードウェアなど）に対する依存性を少なくする。

オープン・ソースやサービスを、アプリケーションから独立して外部調達可能なアーキ

---

[7] http://msdn2.microsoft.com/en-us/architecture/aa699384.aspx

[8] イタリアの経営・社会学者、パレート（V.Pareto）が 1896 年に発表した所得配分の研究に基づく経験則。たとえば、システムの 80%に対する顧客満足度は、システムの 20%に対する顧客満足度によって決まる。

[9] Margaria, T., Service Is in the Eyes of the Beholder, IEEE Computer, November 2007, pp.33-37

テクチャにすることによって、特殊なノーハウや技術をもった小規模企業に成功機会を与え、ロングテール・サービスメニューを次第に豊富にすることができる。これによって、アプリケーション開発・保守事業者は、よりエンタプライズの要求にきめ細かく応えながら、生産性向上、組織合理化を進めることができる。さらに、サービスに、SOA, SaaSで採用される課金方式を適用することが可能になれば、前記、第２の課題に対するひとつの解を手にすることができるようになるだろう。ソフトウェア提供産業は、上記の 2 つの課題に対する解となる「銀メダル」を目指して、goods-dominant（G-D）logicから、service-dominant (S-D) logicを志向し、動き始めている[10]。

第 2 章：　エンタプライズ系ソフトウェアのアーキテクチャ

エンタプライズ系ソフトウェア

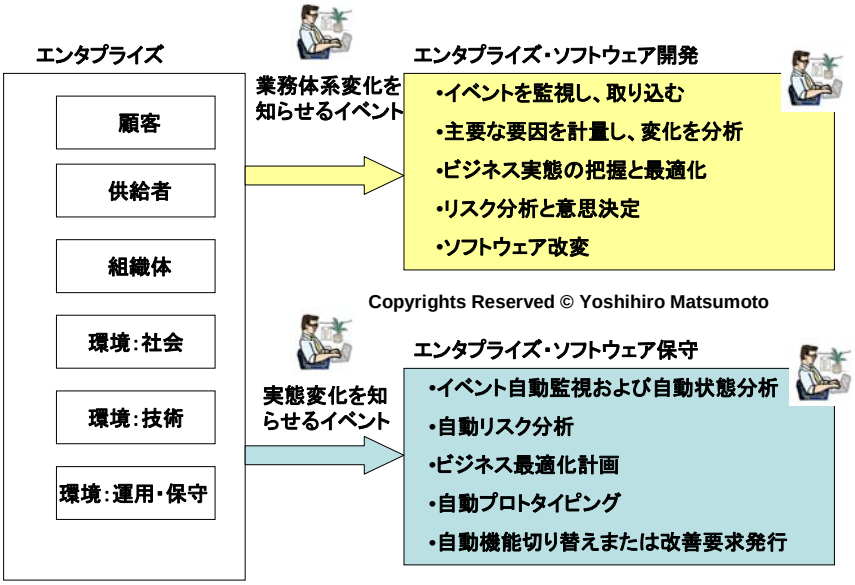
今回は、エンタプライズが何らかの目的、目標をもって導入する ICT (Information and Communication Technology) システムのなかで機能するソフトウェアについて考えてみよう。

エンタプライズとは、公に約束した社会的責任を果たしながら、プロダクトまたはサービスを調達、または提供する契約を遂行するために形成される組織、またはその業務体系を意味する。最近は、提供する側の組織からみた調達組織を顧客 (customer)とよび、顧客の業務体系をエンタプライズ (enterprise)とよんで、両者を区別することが多い。

ICT 専門技術者は、エンタプライズ戦略、戦術および執行を統合したシステム・ソリューション決定に参加し、これを適切に支援する ICT を企画し、ソフトウェア・ソリューションを開発・保守する責任をもっている。図 1 は、エンタプライズ・ソフトウェアの開発および保守に与えるエンタプライズの変化と、それに対して求められる ICT 専門技術者の職務を示している。

[10] http://www.research.ibm.com/journal/sj/471/lusch.html

図 1　エンタプライズの変化とエンタプライズ・ソフトウェアの開発・保守



図の上半は、ソフトウェア開発プロジェクトの実行中の状態、下半は、ソフトウェアがリリーズされて、運用・保守に入った状態を示している。この図が示すように、ICT専門技術者は、開発中はもちろんのこと、運用に入っても常にエンタプライズの変化に対応して、ソフトウェアの保守・改善を行わなければならない。アジャイル開発手法が、これに対するひとつの解法を提供するが、この手法では、開発チームの最低 30%は、A. Cockburn[11] のスキルレベル 2 または 3 をもった人でなければ効果が上がらないとされる。このことは、大規模システム開発・保守では望めない。重要な部分システムに限定して、アジャイル開発手法を適用することが試行されているケースは、IEEE AGILE Conferenceで報告されている。新しいソフトウェアファクトリでは、このようなアジャイルな部分の開発は、前回述べた即発開発、すなわちソフトウェア専門技術者以外のエンタプライズ構成員が直接、手を下せるような開発に委ねることを目標にしている。

サービスとの接続

表 3 は、建築・施工とエンタプライズ系ソフトウェア開発・保守の主なプロセスを対照的に示している。

[11] http://alistair.cockburn.us/index.php/Main_Page

表 3　建築・施工プロセスとエンタプライズ系ソフトウェア開発・保守プロセス

	建築の場合	エンタプライズ系ソフトウェアの場合
計画	顧客から生活・環境・空間の関わりについて要求を聴取し、環境・空間・法令制限などを調査し、予算、工期、構造・材料・環境設備・意匠などの視点から、企画提案書・模型（原形）を作成し、顧客と折衝する。	エンタプライズ戦略およびケーパビリティ分析、エンタプライズ・アーキテクチャ設計
基本設計	ラーメン構造決定、構造強度計算、工法（木造軸組み、ツーバイフォー、鉄骨作り、コンクリート造り、プレハブなど）、基本設計図作成、外観・間取・性能設計	アプリケーション・アーキテクチャおよびデータアーキテクチャ設計
詳細設計	施工設計図作成、構造部材設計、建築材料、建築用資材、インテリア製品、開口部材、照明・音響機器など	テクニカル・アーキテクチャ設計、実装設計、
施工	施工手順書作成、部材、材料、資材、機器等発注、施工管理、検査、承認、清掃、引渡し	プログラム設計、調達（サービス、オープンソフトウェアなど）、プログラミング、テスティング、V&V、実装、システムテスティング、承認、引渡し

建築・施工について注目したいのは、顧客がどのような特殊な要求を出そうとも、要求に対する充足責務は設計者の役割範囲のなかで実現し、構造部材、建築材料、建築用資材、インテリア製品、開口部材、照明・音響機器などは外部から調達し、施工プロセスは工業化され、これも外部調達可能としていることである。

エンタプライズ系ソフトウェアは、「サービス」や「オープン・ソース」を積極的に外部調達し、エンタプライズ・プロセスの変化に応じ、即席開発によってサービスの取捨選択・組み換えができるようにする。エンタプライズ・ソフトウェアフレームワークは、通常、3 層、または 4 層のフレームワークに従って構成される。

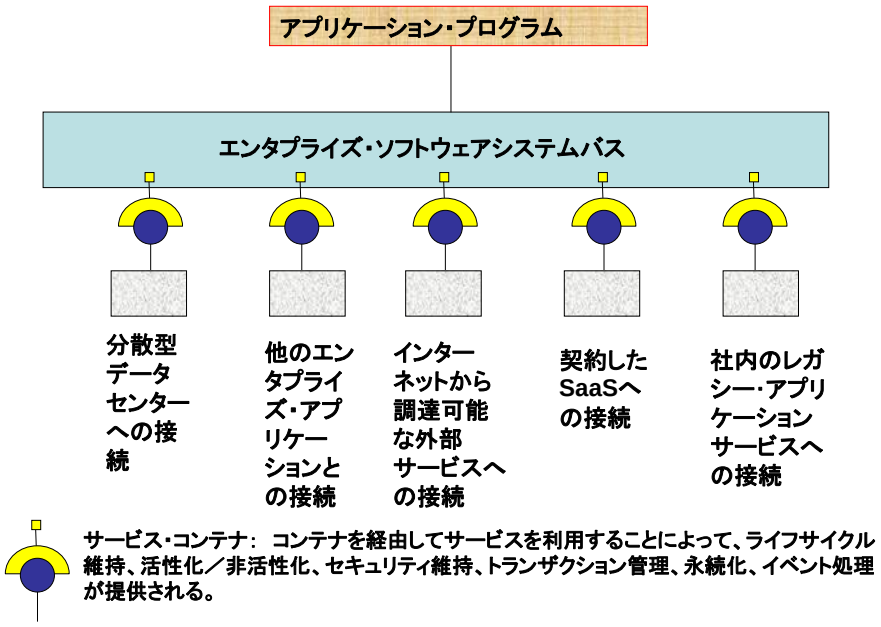
- ユーザ・インタフェース、またはユーザ・エクスペリエンス（user experience）層
- ユーザ・エクスペリエンス支援層
- アプリケーション層
- データ層

ユーザ・エクスペリエンス層というのは、対象とするユーザが、「楽しく」「面白く」「心地よく」ソフトウェアで提供されている機能やサービスを利用できるようにした仮想世界に対するインタフェース（リッチ・クライアント・インタフェースの 1 種）のことをいう。画面のデザイン、レイアウト、画面遷移、参照ドキュメント、手順などを、ユーザの経験に合わせて調整して、提供する。太郎君の外国出張「ユーザ・エクスペリエンス」とは、出張申請書、旅費申請書、外貨購入手続き、航空券予約、ホテル予約、顧客の日程表、帰国後の業務報告書、旅費精算書、およびこれらの関連を形式化した、太郎君の概念世界を

指す。太郎君は、ほかにいくつも定番のビジネスプロセスを実行しているに違いない。太郎君は、定番ビジネスプロセスごとに、ユーザ・エクスペリエンスを、直近、または遠隔でもつことができる。ユーザ・エクスペリエンス支援層では、ユーザ・エクスペリエンス層が複数のユーザにまたがる対話を扱うような場合に、文書やプロセスを一時的に記憶したり、管理を行ったりするための補助的な層である。アプリケーション層およびデータ層は、伝統的な 3 層フレームワークと同じ役割をもつ。

アプリケーション層の一例を、図 2 に示す。この例では、コンテナ[12]を経由して、図の下部に示された 5 種類のサブシステムが、図の左上部に位置するアプリケーション・プログラムのなかで、利用者からの単独、または複合リクエストに応じて組み合わされ、利用される。両者の間を仲介するシステムバスは、非同期メッセージを安全、かつ高信頼に伝達する。このサービスは、図の右上部にある従来型のクライアント・アプリケーションから利用することもできる。

図 2　エンタプライズ・ソフトウェアアーキテクチャの例



外部調達リソースやサービスを安全、かつ高信頼で利用するためには、単位サービスをネットワークから直接取り込むのではなく、図 2 で示したように、コンテナで包んで使う必要がある。コンテナに入れることによって、被調達要素は、ライフサイクル維持、活性化／非活性化、セキュリティ維持、トランザクション管理、永続化、イベント処理が保証される。

12 たとえば、http://www.omg.org/docs/c4i/03-06-08.pdf

太郎君が要求するソリューションには、航空会社の座席予約サービス、ホテルの宿泊予約サービス、訪問する顧客の日程表などへのオンラインアクセスを必要とする。図 2 のなかに示したアプリケーション・プログラムを、太郎君自身による即席開発入力から自動的に生成し、作動し、エンタプライズ・ソフトウェアシステムバスを経由して、これらのサービスを利用できるようにすれば、太郎君の要求に対するひとつの解を形成することになる。

第 3 章：　ソフトウェアファクトリのアーキテクチャ

外部調達可能なサービスの利用

「サービス」の定義は、前に示したとおりである。太郎君の要求を、サービスを用いて構築するとすれば、どのようなものをサービスにすればよいであろうか？順に考えてみよう。

１．旅程作成サービス：　あらかじめ定めた表形式に、太郎君が必要事項を入力（または改訂）すると、自動的にデータ化し、保管し、旅程表を作成し、旅費の仮払い申請書を作成するサービスである。

２．予約マッシュアップサービス：　太郎君の旅程データを使って、インターネット経由で利用できる社外サービス、すなわちフライト座席予約、レンタカー予約、鉄道指定席予約、ホテル宿泊予約などにアクセスし、必要なすべての予約を行い、旅程提案作成、料金計算などを行うサービスである。このようなマッシュアップサービスを、すでに旅行業者などに提供しているベンダーもいるので、それに委託することを考えてもよい。

３．旅程異常処理サービス：　外部で提供されている予約サービスのなかには、異常（たとえば、フライトがキャンセルされたなど）があった場合にそれを予約者に知らせるイベントを公開しているものがある。このようなサービスに対して、イベントへの登録を行う。イベントが発生した場合、それを受けて、影響を受ける太郎君の旅程データを調査し、旅程の改訂提案を複数オプション作成し、太郎君に通知する。太郎君がオプションを選択すると、それに従って、旅程を変更し、影響を受ける予約をキャンセル、または更新する。以上のようなプロセスをまとめて提供するサービスである。

太郎君は、異常処理プロセスを、必ずしも一律に定義できるものではないと考えている。出張のたびに、異常処理プロセスを変更したい。このような要求に応えるために、素人でも描けるような簡単な流れ図によってプロセス定義を画面で作れるようにする。このような手段による開発を、即席開発と呼んでいる。

４．旅行案内サービス：　予約したホテルや、訪問しようとする顧客オフィスの場所を示す地図を、Google 地図サービスなどを用いて作成し、関連する情報を提供するサービスである。

５．出張報告書作成および精算サービス：　帰国後、太郎君が作成する業務報告書および支払い実費報告書様式を提供し、太郎君が記入した後、電子化して、社内の必要部署に送付するサービスである。

これら以外にも、いくつかのサービスが考えられるが、これらサービスが利用するデータは、それぞれが独自のスキーマをもったデータベースで管理されている。また、サービスそれぞれが異なるプロトコルやメッセージ形式を主張する。したがって、メタデータやプロトコル変換などを利用して、上記のサービスを統合する上位サービスを作成する。このようなサービス統合は、SaaS が得意にしているものであり、太郎君のような出張者が大勢いて、頻繁に出張するような企業では、SaaS 業者に開発を委託し、年間使用契約をして利用するという方法も考えられる。

太郎君の要求は、以上のような方法で解決できたとしても、太郎君の会社では、自動化したい様々なビジネスプロセスが存在し、必ずしもSaaSで解決できないものも現れる。そのようなものは、アプリケーション・プログラムを開発して対応しなければならない。しかし、このアプリケーション・プログラムは、様々な外部調達サービスやSaaSを利用しながら実行しなければならない。これを可能にするソフトウェアアーキテクチャを、マイクロソフト社は、S+S (Software plus Service)[13] と呼んでいる。S+Sの最終版提供は、まだ行われていない。

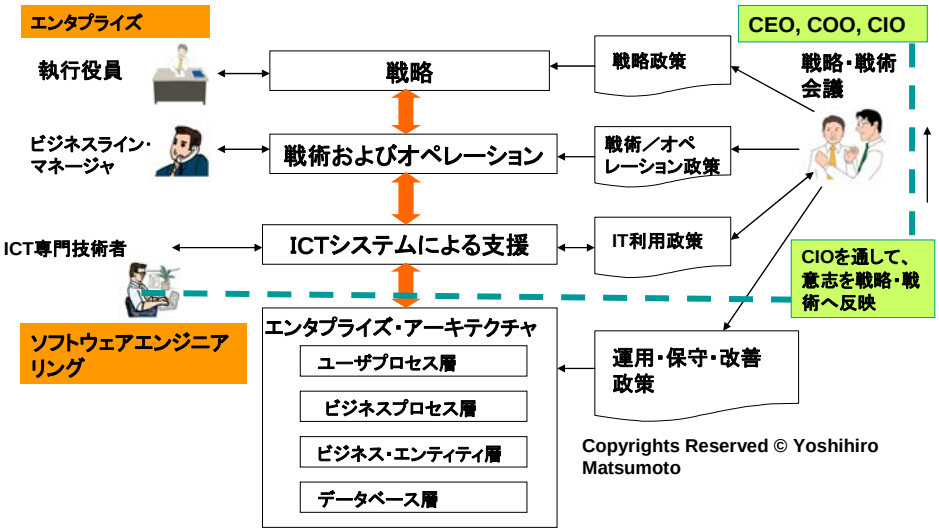
ICT 専門技術者の役割

エンタプライズのなかでの ICT 専門技術者（ICT professionals）の活動領域は、2 つに分かれる。ひとつは、図 3 に示す「エンタプライズ領域」、いまひとつは「ソフトウェアエンジニアリング領域」である。専門技術者の重要な役割は、この両領域をシームレスに編み合わせることにある。

ICT 専門技術者の第 1 の役割は、エンタプライズ・ソリューションを文書化し、執行役員を含めたエンタプライズ戦略および執行責任者の意思決定を促すことにある。決定された内容は、ICT 専門技術者によってソフトウェアエンジアリング技術者に伝達され、ソフトウェア・ソリューションに対する要求となる。ICT 専門技術者の第 2 の役割は、ソフトウェア・ソリューションの実現・実装・保守に対する執行責務である。図 1 で説明したように、ソフトウェア・ソリューションは、エンタプライズ・ソリューションのダイナミズムに整合、適応、即応しなければならない。

[13] http://msdn2.microsoft.com/en-us/architecture/aa699384.aspx

図 3　ICT 専門技術者の役割と活動領域



ICT専門技術者は、ビジネスライン責任者によって決定されたエンタプライズ・ソリューションを、「プロセス・パック」の連携に展開する[14]。たとえば、太郎君の外国出張に必要な前記のビジネスプロセスは、ある数以上の社員が共通に必要とするものであると判断され、ひとつの社内イントラネットサービスにしたいと決定が下りたとき、ICT専門技術者は、この外国出張業務支援プロセスをひとつのプロセス・パックとして定義する。プロセス・パックは、対象とするビジネスプロセスを定義したメタデータ、実現に必要なソフトウェアコンポーネント（外部から調達可能なサービスを含む）、およびコンポーネントの結合を示すテンプレート、プロセス・パックに含まれるアクティビティとアクティビティ・フロー、および他のプロセス・パックに対するインタフェースから成る。プロセス・パックは、エンタプライズ・ソリューションを構成する単位ビジネスプロセスをソフトウェアサービスとして資産化し、エンタプライズ・プラットフォームに登録するための開発仕様書のようなものである。

ソフトウェア・セル生産

筆者は、ひとつのプロセス・パックを開発するソフトウェアファクトリチームのことを、ソフトウェア・セル[15]とよび、ソフトウェア・セルの連携によって、ひとつのソフトウェア・

[14] http://msdn2.microsoft.com/en-us/office/aa905528.aspx
[15] http://www5d.biglobe.ne.jp/~y-h-m/IntroductionToSF.pdf

ソリューションを開発・保守する体制を、ソフトウェア・セル生産と呼んでいる。たとえば、火力／原子力発電システム開発プロジェクトのなかのひとつのセルで、原子炉、蒸気タービン、発電機の性能をオンラインで計算・分析し、プラント運転を最適にガイドするサービスを開発しているとする。開発されたサービス資産を、たとえば、Microsoft BizTalk サーバのようなものに実装し、エンタプライズ・プラットフォームに接続すれば、ひとつの原子力発電所内だけでなく、エンタプライズ・プラットフォームに結合された社内各事業所から、オンライン・サービスとして利用できるようになる。

セルのなかで開発・保守されるプロダクト、またはサービスは、大きく分けて 3 つの種類に分かれる。

- ✓ インターネット上で公開されている調達可能なサービスをそのまま組み合わせて、エンタプライズメンバに提供するプロダクト、またはサービス (building-block services)：たとえば、太郎君の外国出張予定に従って、航空機座席予約およびホテル宿泊予約をする。
- ✓ 上記と同様、外部調達可能なサービスを組み合わせるが、その組み合わせが複雑、セキュリティ（個人認証、権利認証、個人情報保護）の保証が必要、即席開発機能の組込みが必要、などの価値付加が要求されるプロダクト、またはサービス：たとえば、太郎君の社内報告、経費精算、訪問先顧客の日程表へのアクセス、旅程変更のための即席開発機能などを処理するプログラム、またはサービス (value-added services)。
- ✓ 外部調達サービスをまったく利用せず、社外秘としてエンタプライズメンバに提供する特別なプロダクト、またはサービス (proprietary services)。

大規模なエンタプライズ・ソリューションになると、ひとつのプロセス・パックの開発に、異なる役割をもったチームが複数参加しなければならないことになる。International Council on Systems Engineering (INCOSE)[16]は、ソフトウェアファクトリのなかで活動するチーム、およびその役割をまとめたINCOSE Handbook v.3.1 を、国際標準化機構 (ISO) に対して、新規作業項目のひとつとして提案しようとしている。ここでは、4 種類のチーム（ビジネス開発、システム統合、ソフトウェア統合、ソフトウェア開発)、および全体を統括するスタッフ組織の存在が示されている。チームの役割は、EXTERNAL, SYSTEM, ELEMENT, SUBSYSTEM, COMPONENT, SUBASSEMBLY およびPARTの 7 レベルに分類され、それぞれのチームがカバーすべきレベル、および担当すべき責務が示されている。大きなプロセス・パックの開発では、この 4 種類のチームすべてが参加する。

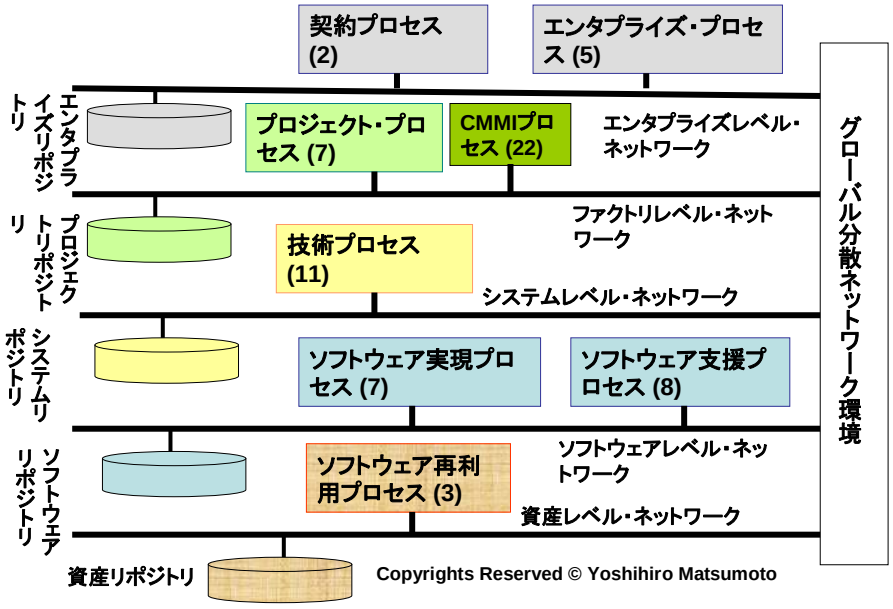
ソフトウェアファクトリ・アーキテクチャ

これら 4 種類のチームおよびスタッフ組織の活動を支援するソフトウェアファクトリ・アーキテクチャの一例を、図 4 に示す。5 層のネットワークは、それぞれ、上から順に、ビジネス開発チーム、システム統合チーム、ソフトウェア統合チーム、ソフトウェア開発チ

[16] http://www.incose.org/

ームおよびスタッフ組織の活動を支援する。これらネットワークは、グローバルに展開される分散ネットワークに接続され、オフショア・アウトソーシングに対しても利用できるようにする。図 4 には、具体的に支援するプロセス集合名と、それぞれのプロセス集合を構成するプロセス型の数をカッコ内に記載している。具体的なプロセス型は、紙面の制約があり列挙できないが、ISO/IEC FCD 12207 Software Life Cycle Processes、ISO/IEC 15288-2002 System Life Cycle ProcessesおよびCMMI v.1.2[17]のなかで明示されている。

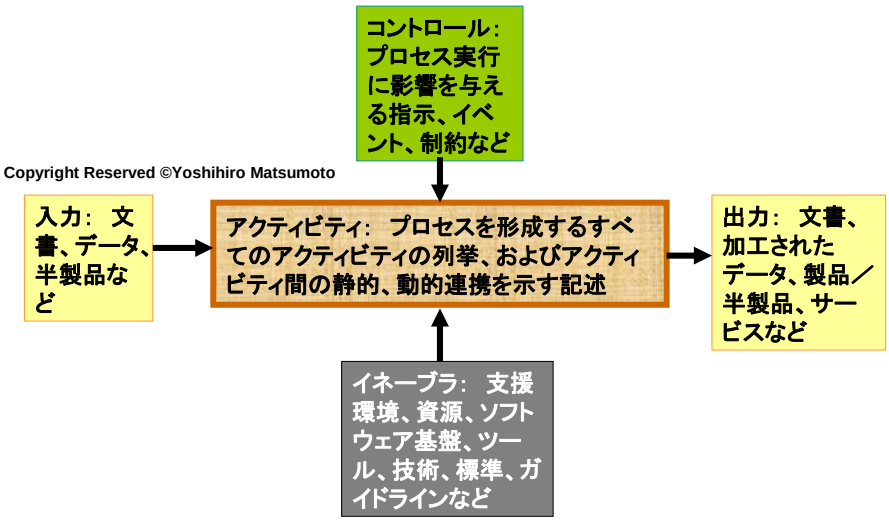
図 4　システム開発・保守のためのソフトウェアファクトリ・アーキテクチャ



個々のプロセス型は、それが属するプロセス集合によってそれぞれ異なる性質をもっている。ソフトウェアファクトリでは、すべての開発・保守および管理・制御活動、資源、製品／半製品を、図 5 に示す「プロセス鋳型」のなかに入れて、マネージする。プロセス鋳型は、5 つの要素（アクティビティ、入力、出力、コントロール、イネーブラ）から構成され、各要素のもつ役割は、図 5 に示すとおりである。プロセス鋳型の各構成要素のなかに入れるべき具体的な内容物は、INCOSE Handbook v.3.1 のなかで列挙されている。

[17] http://www.sei.cmu.edu/cmmi/translations/japanese/models/

図 5　プロセス鋳型



ここで話題を転じ、料理の「レシピ」について考えてみよう。レシピは、構造的、かつ動的に組み合わされたアクティビティから構成されるプロセスである。同じ仕様の料理に対するレシピは、レシピに対するコントロールとイネーブラによって決まる。

ソフトウェア開発・保守プロセスは、レシピに似た性質をもっている。ソフトウェアファクトリ構築の第 1 ステップは、共通の特徴をもったアプリケーションを識別すること、第 2 ステップは、識別されたアプリケーション開発・保守に対するコントロールとイネーブラを特定し、レシピに相当するプロセスの可塑化を行うことである。可塑化とは、アプリケーションに必要なプロセスを、ISO/IEC FCD 12207 Software Life Cycle Processes、ISO/IEC 15288-2002 System Life Cycle Processes および CMMI v.1.2 のなかから選択し、固定型（fixed type）プロセスとバライアント型（variant type）プロセスに分け、プロセス鋳型を使ってそれぞれを整形し、固定型、バライアント型を使い分けながら、必要なプロセスが互いに連携するようなトポロジ（ネットワーク形態）を作ることである。可塑化は、共通な特徴をもったアプリケーション領域における、ある程度の年月をかけて経験を積み重ねた結果に基づいて形成される。このことを、マカロニ・グラタンを調理するレシピに例えて説明しよう。

✓　第 1 ステップ：　マカロニ・グラタンというカテゴリがカバーするメニューの範囲を

決める。

- ✓ 第 2 ステップ：　マカロニ･グラタン調理に対する制約、すなわち調理人の数、調理設備の容量、注文を受けてから料理を顧客に提供するまでの最長許容時間などから、必要なプロセス（たとえば、材料を揃える、マカロニを煮る、具を作る、スープを作る、オーブンで焼く）を定義し、プロセスライン[18]（プロセスの連携を示す典型的なトポロジのこと）を定義する。マカロニ・グラタンには、パスタや具を変化させることによって、いくつかの種類が提供される。したがって、プロダクトライン国際会議では、可変要素をバリアブルズ（variables)とよび、プロセスのなかの可変部を挟む分岐点および合流点を、バラアビリティ・ポイント（variability point）と呼んでいる。可変部を内包するプロセスがバライアント型、内包しないプロセスが固定型である。レシピ、すなわちプロセスラインを資産化し、再利用可能にするために、個々のプロセスの特性を、プロセス鋳型を用いて定義する。

世の中には、調理ガイドブックというものがある。このなかには、いろいろなレシピが、適切に分類されて集積されている。プロセスラインとは、アプリケーション型（メニューに相当）および制約（コストおよび開発期間など）をパラメータにしてクラス分けされた、ライフサイクルプロセス・ガイドブックを意味する。調理ガイドブックに倣って、ライフサイクルプロセス・ガイドブックに記載する個々のプロセスは、その形式を揃えて読み易くするため、図 5 で示したプロセス鋳型に従って記述される。70-80 年代の日本のソフトウェアファクトリでは、このようなライフサイクルプロセス・ガイドブックを現場で整備して、実務の効率化を図っていた。

第 4 章：　システム系ソフトウェアの開発･保守

プロダクトライン

宇宙機器、防衛機器、輸送機器（カーナビを除く自動車を含む)、発電機器（原子炉、ボイラを含む)、環境システム機器、プラント計装・制御機器、ロボティクスなどに組込むソフトウェアの大部分は、エンタプライズ系とは異なり、自然法則、工学的定理・方法論などの適用に基づく、実時間における正しさの検証および確認（verification and validation）を必要とする。現代社会は、エンタプライズ系に夢中であるが、資源が枯渇し、エネルギー問題、環境問題が社会を揺るがすようになれば、ここでいう「システム」がエンタプライズにとって代って中心課題になるであろう。以下、これら「システム」の中心的役割を担うソフトウェアを、ISO/IEC 15288-2002 System Life Cycle Processes に倣って、システム系ソフトウェアとよぶ。組込み系とよばれるソフトウェアは、このなかに含まれる。

一般に、システム系アプリケーションでは、共通の性質をもつアプリケーション領域を

[18] http://www5d.biglobe.ne.jp/~y-h-m/Processlines&Productlines.pdf

特定して分類・体系化されたソフトウェア部品、ツール、ガイドラインなどを記憶・管理したプロダクトライン、またはプロダクトファミリを利用した開発・保守が行われる。

プロダクトライン（product line)、およびプロダクトファミリ（product family)は、ソフトウェアプロダクトの特徴を識別し、特徴を体系化し、この体系に従って分類された再利用可能なソフトウェア資産の体系を意味する。両者の違いは、前者が利用者の視点からみた特徴に基づき、後者は製作者の視点からみた特徴に基づいていることである。しかし、両者を区別することによる効果は、あまり見られない。

歴史的にみて、プロダクトファミリは、1995年からスタートしたEUのEspritプロジェクト「ARES (Architectural Reasoning for Embedded Systems)」のなかで鋳造された概念であり、主として欧州で進められていた[19]。プロダクトラインは、同じころから、Carnegie Mellon University/ Software Engineering Institute[20]で進められた概念で、マクドナルド社販売店のなかにある調理プロセスラインからヒントを得たとされている。両者は、上で述べたように視点が異なっていたが、2005年からは、「プロダクトライン」という名称に統一し、議論されるようになった。

システム系ソフトウェアは、整備・管理・保守が適切に行われているプロダクトライン、またはプロダクトファミリを利用して、開発・保守される。

システム系ソフトウェアの開発方式

60 年代 90 年代まで、システム系ソフトウェアでは、実時間性保証などの視点から、プラットフォーム（OS、言語、通信基盤、ハードウェアなど）を含めた垂直統合型開発・保守が行われてきた。サイロ型開発・保守ともよばれる。ところが、90 年代に入ってから、水平分業による雇用拡大を可能にする開放型共通プラットフォームがグローバルに採用されるようになり、特定の製作者によるプラットフォームの寡占化が進むようになった。

システム系ソフトウェアのなかで、その品質特性（ISO/IEC 9126, JIS X 0129）がプラットフォームに依存するような実時間アプリケーションでは、垂直統合型開発・保守が原則とされてきたが、最近は、グローバルに共通化されているプラットフォームを外部から調達して利用するケースも見られる。しかし、とくにミッション・クリティカルなシステムでは、調達元における随時の変更や改訂をコントロールできないことが致命傷になることがある。したがって、アプリケーション・ソフトウェアからみたプラットフォームに対する依存性を低減するため、前者に対して後者を仮想化するための「フレームワーク」を、プラットフォームの上に被せる。図 6 は、システム系ソフトウェア開発・保守における大雑把なアーティファクト（半製品およびツール）の関連を示している。

---

[19] F. van der Linden: Software Product Families in Europe: The Esaps & Café Projects, IEEE Software, July/August 2002, pp.41-49

[20] http://www.sei.cmu.edu/

図 6　システム系ソフトウェア開発・保守におけるアーティファクト関連図

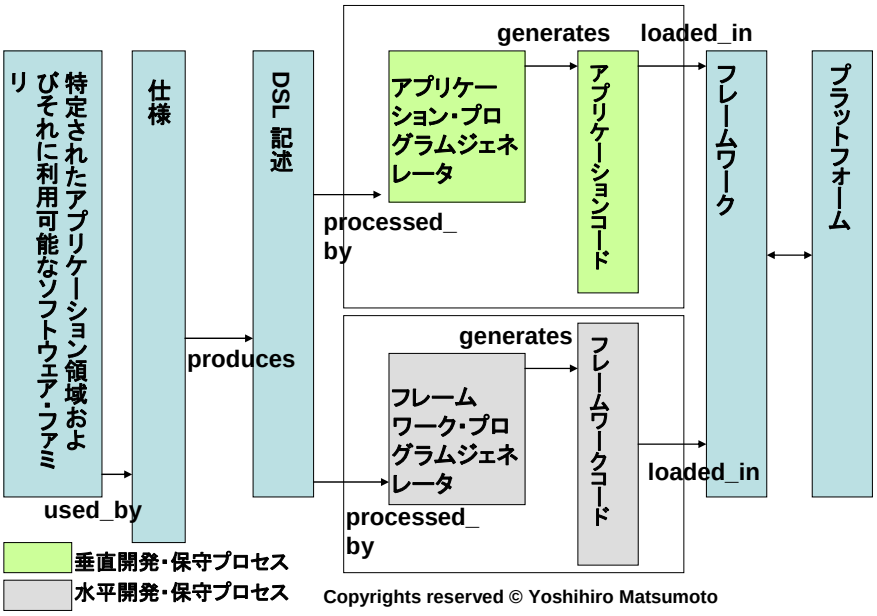
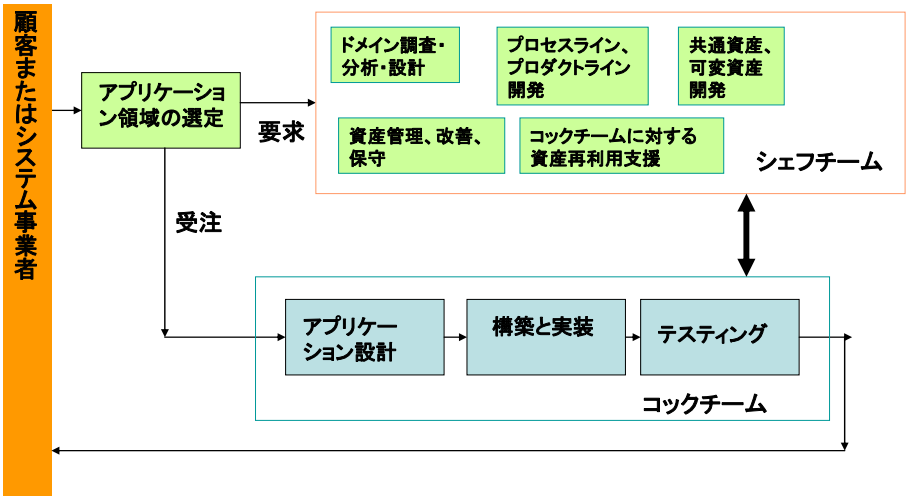


図 6 のなかで示す DSL (Domain-specific Language)、アプリケーション・ジェネレータ、フレームワークジェネレータ、フレームワークは、特定されたアプリケーション領域に対してあらかじめ開発される。この開発は、ラインと切り離した別チームで行われる。外食サービスの場合、飲食店は、一般に懐石料理、フランス料理などというようにアプリケーション領域を特定して設営される。そこには、店舗を特徴付けるメニューがあり、それはシェフによって開発される。メニューに従って注文された料理は、コックが調理する。ソフトウェアファクトリは、これに倣って、便宜上、上で述べた開発を行うチームのことをシェフチームとよび、顧客からの注文に応じてシェフチームが開発した DSL、コードジェネレータ、フレームワークなどを利用してプロダクトを開発するプロジェクトを、コックチームと呼んでいる。

図 7　シェフチームとコックチームの役割



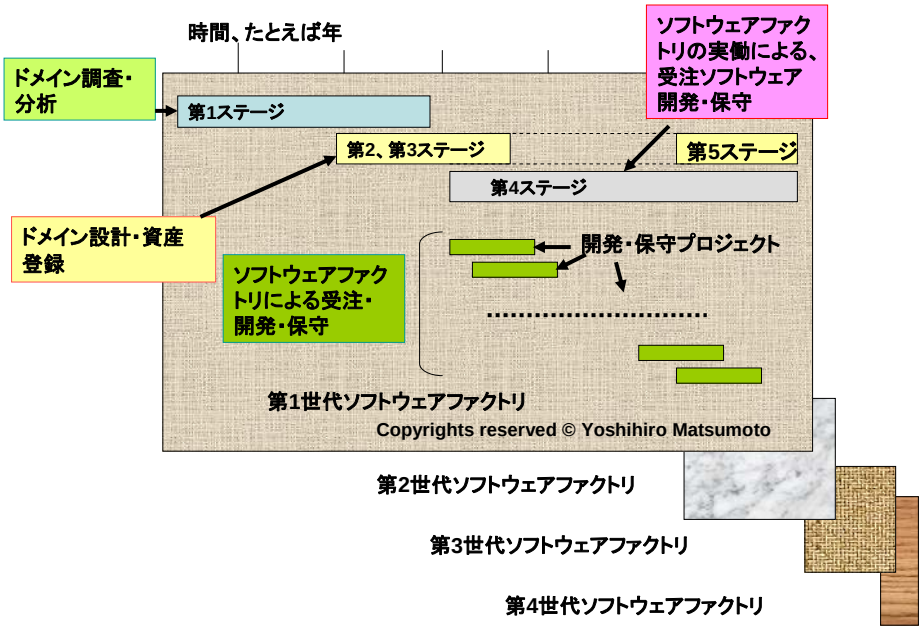
システムは、エンタプライズと違って、物理系システムと人間系システムから構成される。物理系システムは、自然法則を基にして発明され、体系化された定理を応用して構築された人工的機構である。人間系はそれを操作し、それの作用を利用する人、または人の組織である。物理系システムは状態をもち、人の操作および物理系システムのさらに外にある世界からの刺激を受けて、状態を推移する有限状態機械として定義することができる。人間系システムは、この有限状態機械と交信して、状態推移をリアルタイムにコントロールし、それぞれの状態が生成する作用を利用して、必要とする効果と利益を得る。

第 5 章：　ソフトウェアファクトリの構築と実践

今回は、システム系ソフトウェアファクトリの構築と実践について述べる。ここでもっとも重要なことは、システム系ソフトウェア開発・保守事業経営責任者が、ソフトウェアファクトリを十分理解し、過去に蓄積された実績を駆使できるソフトウェアファクトリを構築・実践すれば、十分な成功ビジネスを期待できそうなアプリケーション領域があることに気づくことである。ソフトウェアファクトリのライフサイクルは、この段階から始動する。ソフトウェアファクトリは、図 8 に示すライフサイクルをもち、ひとつのライフサイクルは、つぎに示すステージの連接で構成される。各ステージのなかで実施するプロセ

スは ISO/IEC FCD 12207 Software Life Cycle Processes に準拠しており、国際標準を遵守することによって、ソフトウェアファクトリの国際化、たとえばコックチームのオフショア・アウトソーシングが可能となる。この説明では、前回の図 7 をあわせて参照されたい。

図 8　ソフトウェアファクトリのライフサイクル



- ✓ 第 1 ステージ（ドメイン調査・分析）：　シェフチームは、かなりの年月をかけて、すでに製品出荷が終わった複数のプロジェクトの開発プロセスを同定・計量し、開発されたアーティファクトを分析することによって、共通性をもったプロジェクト群を識別し、これからアプリケーション領域の範囲決め（scoping）を行い、その結果をドメインとする。
- ✓ 第 2 ステージ（ドメイン設計）：　シェフチームは、ドメインのもつ特徴（features）、能力（capabilities）、概念（concepts）、機能（functions）、および開発･保守プロセスを識別し、それらを共通（固定）部分と可変部分に分けて体系的に示す「ドメインアーキテクチャ」を作成する。つぎにドメインアーキテクチャを実現するためのソリューションを設計し、ソリューションを実現するために必要な構成要素を、過去の蓄積遺産を利用して開発し、それらを再利用できるように加工し、資産化する。
- ✓ 第 3 ステージ（資産登録 asset provision）：　シェフチームは、機能要求をパラメータとするプロダクトライン（またはプロダクトファミリ）を構築する。マクドナルド社販売店の調理プロセスラインのように、標準化できるプロセスの連携スキーマが形成

できる場合には、非機能要求をパラメータとするプロセスラインも構築する。

- ✓ 第 4 ステージ（ソフトウェアファクトリの実働、コックチームによる受注開発・保守、シェフチームによるコックチームに対する資産提供、資産利用支援、資産管理・改善・保守）：　この段階から、コックチームは、新しい受注を、登録された資産を利用して開発・保守するようになる。このステージでは、シェフチームは、コックチームに対する資産利用指導・支援を行うとともに、資産再利用率を常に監視し、再利用率が期待値に達しない場合には、資産の増補・改善･保守を行う。
- ✓ 第 5 ステージ（ライフサイクル更改処理）：　一般にシステム系プラットフォームは、技術革新が激しいため、短い周期で変化する。また、アプリケーション開発技術の改廃も頻繁に行われる。したがって、ソフトウェアファクトリも、ある期間（通常は 5 年以上、10 年未満）を経ると、全面的に見直す必要が起きる。ライフサイクルの終期においては、現存のソフトウェアファクトリを運用しながら、第 1 ステージに戻って資産およびファクトリ・マネジメントを見直し、新しいライフサイクルへ向けての準備を行い、切り替えを実行する。

第 4 ステージ以降のソフトウェアファクトリ運用に当たっては、常に投資回収率を監視し、第 4 ステージで得るキャシュフローによって、第 1 から第 3 ステージまでに投じた初期投資を回収するようにしなければならない。具体的な方法についてはいくつかの提案がされている[21]。

以上 5 回に亘って、ソフトウェアファクトリについて解説したが、つぎの言葉をもって終わりとする。

- ・ ソフトウェアファクトリは、アプリケーション領域、プロダクトおよびプロセスがもつ共通性を洞察し、それを抽出して可視化し、再利用可能にし、事業化する能力をもった事業家の存在によって成功する。
- ・ ソフトウェアファクトリでは、3 つの役割、すなわちシェフの役割、コックの役割、素材提供者の役割を区別してマネージすることが必要である。
- ・ 優れたシェフを確保・育成すること、品質・生産性意欲をもつコックを育てること、優れた素材提供者の協力を幅広く求めること、が成功への鍵である。
- ・ シェフによってコモディティ化されたプロダクト（コックが開発・保守できるようにプロセスが定番化されたプロダクト）は、できるだけサービス化する。
- ・ コック、素材提供者は、（オフショア）アウトソーシングしてもよいが、シェフをアウトソーシングしてはならない。

以上

[21] たとえば、Matsumoto, Y.: A Guided for Management and Financial Controls of Product Lines, Proc. SPLC2007, pp.163-170