

ソフトウェア・セル生産方式の概要

松本 吉弘
京都高度技術研究所・顧問
(元京都大学教授)

■ 背景 (context)

ビジネスアーキテクチャ論から、ソフトウェア生産ビジネスアーキテクチャのあり方を考えてみよう。

■ とり上げる問題 (problem)

日本的組織文化のなかから生まれたセル生産方式を、ソフトウェア生産に適用する方法を考えて見よう。

■ 解 (solution)

ソフトウェア・セル生産方式の実践方法を考えてみよう。

背景

**ビジネスアーキテクチャ論から、ソフトウェア
生産ビジネスアーキテクチャのあり方を考
えてみよう。**

ソフトウェアファクトリの位置づけ

藤本隆宏著：日本のもの造り哲学、日本経済新聞社刊(2004)、図2 (p.43)を用いて説明

事業の形態	もの造りの組織能力	深層のパフォーマンス	表層のパフォーマンス	収益力
システム事業会社	領域別に分社化し、システム機器およびソフトウェアを一貫生産	ソフトウェアファクトリ	本社システム営業および営業技術によってブランド化を推進	特定領域システム製品を多売し、投資回収
レストラン	特定カテゴリに特化し、一流シェフによるコック組織編成	厨房組織	料理メニューおよび接客組織	秀逸メニュー料理を多売し、投資回収

事業領域は： 国益、公共、情報通信、企業経営・産業、社会・環境、教育・科学・技術等を適用対象とするシステム

行列のできるレストランは

- レストランは、特定ドメイン(例:懐石料理、フランス料理)に、
スコープを限定している。
- シェフは、世界的名声を保有している。
- シェフとコックの職域を明確化している。
 - シェフは、メニュー項目とそれに対応する料理を開発
 - シェフは、各メニューに対する料理仕様、レシピ、材料を定義
 - コックは、客席からの注文を受け、シェフの定義に沿って調理
 - シェフは、コックを育成、指導、監督
 - シェフは、対顧客責任を全うする。

行列のできるソフトウェアファクトリは

レストランのシェフモード	ソフトウェアファクトリでは、イノベーションモード
レストランのコックモード	ソフトウェアファクトリでは、コモディティモード

- マーケティング・スコープを、限定されたアプリケーション・ドメインにとどめている。
- プロダクトメニューを定義している。
- メニュープロダクトの開発(イノベーションモード)には、思い切って投資している。
- イノベーションモード開発は、通常、インテグラル・プロセスで行っている。
- イノベーションモードで開発したプロダクトを初出荷後、インテグラル・プロセスをモジュラ・プロセスへ変換し、アーティファクトおよびマイクロ・プロセスをデータベース(プロダクトライン)化し、コモディティモードへ移行する。
- イノベーションモードで開発したメニューに従って、新しい注文をどっさり獲注する。
- イノベーションモードで投資した開発資金を回収し、高い収益性を目指している。
- イノベーション開発のためのリードタイムの長さが短い企業ほど、すぐれた深層パフォーマンスをもつ。

ソフトウェアエンジニアリングにおけるパラダイム回帰(1)

1. 一般性への反逆

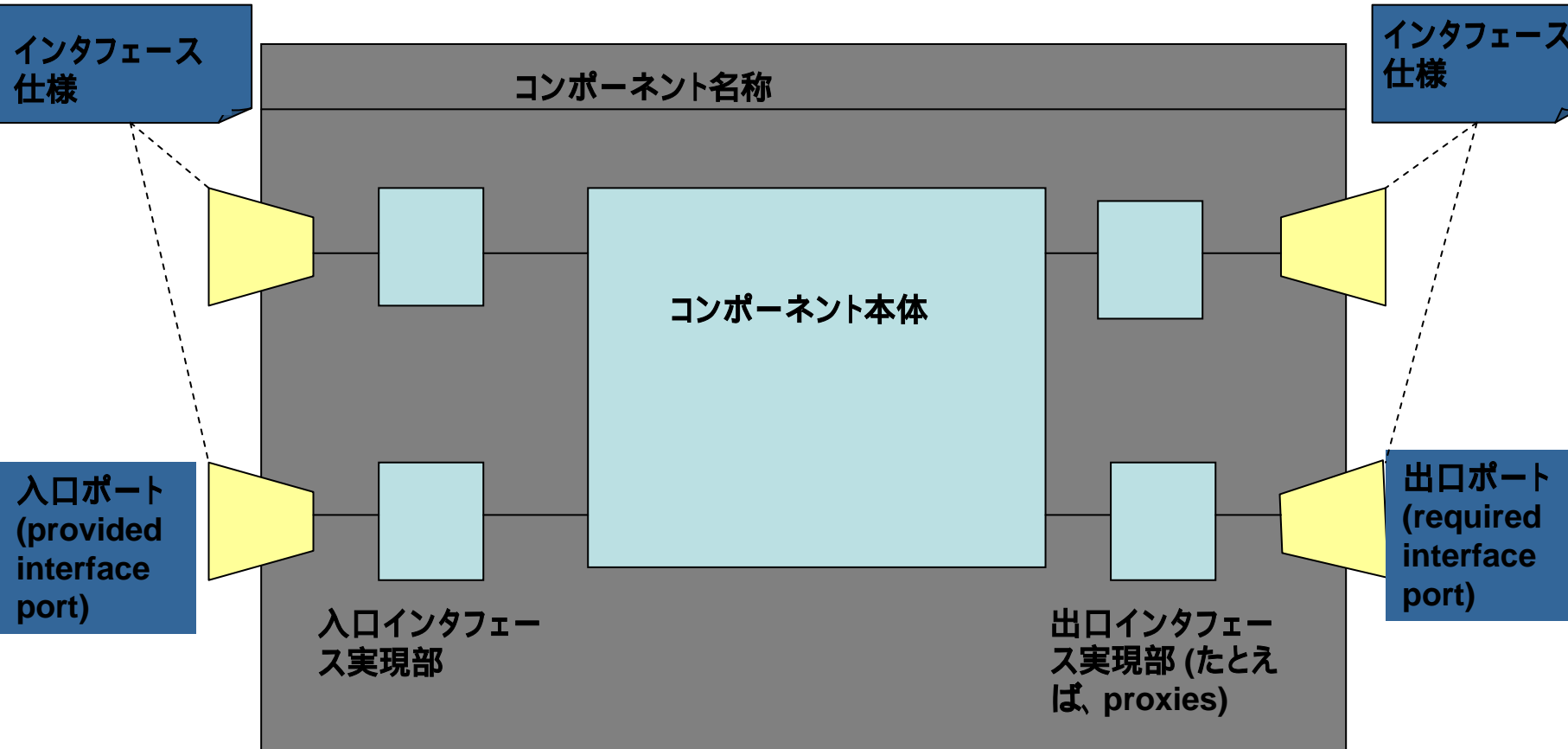
90年代のパラダイム	再帰する日本的パラダイム
プロダクトおよびプロセスに関する、「実質が乏しい一般性 (gratuitous generality)」を追求し、ときには、このような一般性に従った観念的エンジニアリング・プラクティス(実践)を、技術者に強制してきた。	ある高さの抽象レベルまでは一般性を追求するが、それより低い抽象レベルに関しては、それぞれのアプリケーションで与えられる要求と制約に適したエンジニアリング・プラクティス(実践)を展開する。

ソフトウェアエンジニアリングにおけるパラダイム回帰(2)

2. ソフトウェアプロダクトの部品化

90年代のパラダイム	再帰する日本的パラダイム
一石型 (monolithic)パラダイム: たとえば、オブジェクトでは、入口インタフェースは定義されるが、出口インタフェースは定義されず、入口、出口とも、コントラクトは定義されていないため、部品として扱えない。	サービスコンポーネントからプログラムモジュールに至るまで、入口 / 出口インタフェースおよびコントラクト定義を与え、コンポーネント間接続、およびマザーボードとなるソフトウェア共通基盤へのコンポーネントの差し込みを行うことによって、システムを組み立てる。

コンポーネント



ソフトウェアエンジニアリングにおけるパラダイム回帰(3)

3. 開発の類似性への着目

90年代のパラダイム	再帰する日本的パラダイム
とくにオープンシステムになってから、アプリケーション開発では、One-Off development (一回きりの開発)と呼ばれる方式が一般的となった。ひとつの開発が終われば、そこで形成された資産が再利用されることがきわめて少なかった。	プロダクト、プロセス、人の領域知識 / スキルを系列化して、組織的に再利用する。なぜ日本の自動車産業は成功しているか、を考える必要がある。

ソフトウェアエンジニアリングにおけるパラダイム回帰(4)

4. ビジネス(システム)とソフトウェア構築 (construction)の直結

90年代のパラダイム	再帰する日本的パラダイム
要求、基本設計、詳細設計プロセスを独立したものとして識別し、中間的な仕様を作成して、コンベヤー式に委託開発することが慣例とされてきた。	ビジネスモデルから直接コードを自動生成し、ビジネスイノベーションを計画する段階で、即時プロトタイピングを行い、ビジネスとプログラムの間を直結してシステム化する。

- **ビジネス変化への即応**

- **intrinsic (functional) な要求とaspectual (non-functional) な要求をそれぞれ分離独立して実現し、NOV (net option value)を考慮しながら、個別要求要素をコンポーネント化して編み合わせる (weave) するアーキテクチャとする。**
- **コモディティ・ソフトウェアの生産は工業化(自動化)し、ソフトウェア・サプライズ (software surprise: ソフトウェアの不可解性)に対するコンピテンスをもった人材は、ビジネスエンドに密着して開発させるようにする(この業務はアウト/オフショア・ソーシングしない)。**

アーキテクチャの定義

- アーキテクチャは、高い抽象水準における設計思想の表現である。
- アーキテクチャは、システムの基礎的構造を示すために用いられる。
- アーキテクチャは、プログラムまたはシステムにおける(1)機能要素の組み立て、(2)機能要素間の関係、(3)設計および将来の進化を方向付ける原則および規則を示すために用いられる。
- アーキテクチャには、機能要素(components)および接続要素(connectors)が示されていないなければならない。
- IEEE Std 1471-2000, IEEE Recommended Practice for Architectural Description of Software Intensive Systems

インテグラル・アーキテクチャ (integral architecture)と モジュラ・アーキテクチャ (modular architecture)

- プロダクトの機能が複数の構成部品にまたがって複雑に配分され、部品間の相互依存性および相互操作性が明確にできていないようなアーキテクチャのことを、インテグラル・アーキテクチャとよぶ。
- プロダクトの単位機能が、部品に対して1対1、ないし集合論でいう「マッピング:写像」関係として定義することが可能であり、相互依存インタフェースおよび相互操作コントラクトが事前に明示されているようなアーキテクチャのことを、モジュラ・アーキテクチャとよぶ。

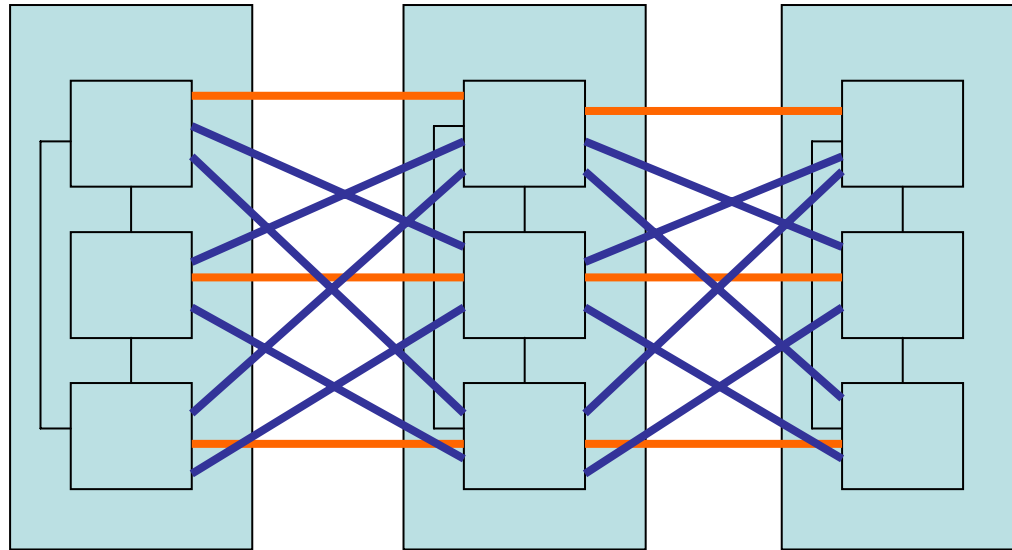
参考文献 藤本隆宏ほか: ビジネスアーキテクチャ、有斐閣 (2001); Baldwin, C.Y., and K.B. Clark, Design Rules: The Power of Modularity, MIT Press (2000)

アーキテクチャのインテグラル性とモジュラ性

概念レベル

論理レベル

物理レベル



デザイン・アーキ
テクチャ

- デザイン・アーキテクチャが、レベル間において、モジュラ性を有する場合、オレンジ色で示した依存性をもつ。
- デザイン・アーキテクチャが、レベル間において、インテグラル性を有する場合、青色で示した依存性をもつ。

設計論から

- **設計とは： 機能概念集合を、属性概念集合へ写像する関数を定義する行為である。**
- **機能概念は、無定義な用語であり、ときとして、挙動と状態の組として表すことができる。**
- **属性概念は、人工物に付帯し、あらかじめ人によって認知されている概念である。**
- **機能概念集合族、および属性概念集合族は、それぞれ異なる位相空間に属しているが、共通台集合をもつ。一般に、両位相空間の関係は、メタモデルによって表わされる。**
- **設計は、しばしばアブダクション(ある個別の事象を、もっとも適切に説明し得る仮説を導出する行為)によって遂行される。**

- つぎの3つの抽象レベルに分けて、設計する：
 - 概念レベル
 - 機能概念集合を分析し、位相空間に展開し、表現する。
 - 表現文書の例： 企業情報モデル、ビジネス・プロセスモデル、アクティビティ対情報・対照モデル、ビジネス・イベントリスト、ビジネス構造モデル、ロール対アクティビティ・対照モデル
 - 論理レベル
 - 機能概念を、属性概念に写像し、属性概念を位相空間に展開し、表現する。
 - 表現文書の例： ユースケース、ユーザインタフェース・デザイン、ユーザプロセス (ConOps)、アспект、クラス、ビジネスプロセス、シーケンス、コラボレーション、状態推移モデル、データベース・スキーマ、CRUD
 - 物理レベル
 - 属性概念を、実装概念 (プログラム、API、IDLなど) に写像する。

インテグラル性とモジュラ性に関して

- シェフモードは、原則としてインテグラル・アーキテクチャを主体にして行う。
- コックモードは、原則としてモジュラ・アーキテクチャを主体にして行う。

- 通常、シェフモードで行うイノベータイプなデザインは、インテグラル・アーキテクチャ
- デザインをコモディティ化する過程において、インテグラル・アーキテクチャをモジュラ・アーキテクチャへ変換する努力が行われる。
- 変換のための常設組織を確立している企業が勝つ。
- 変換において、プロダクトグリッドから、プロダクトラインへの変換が行われる。
- この変換機構が確立されていることが、1970年代日本企業の特徴であった。

ソフトウェア・セル生産では

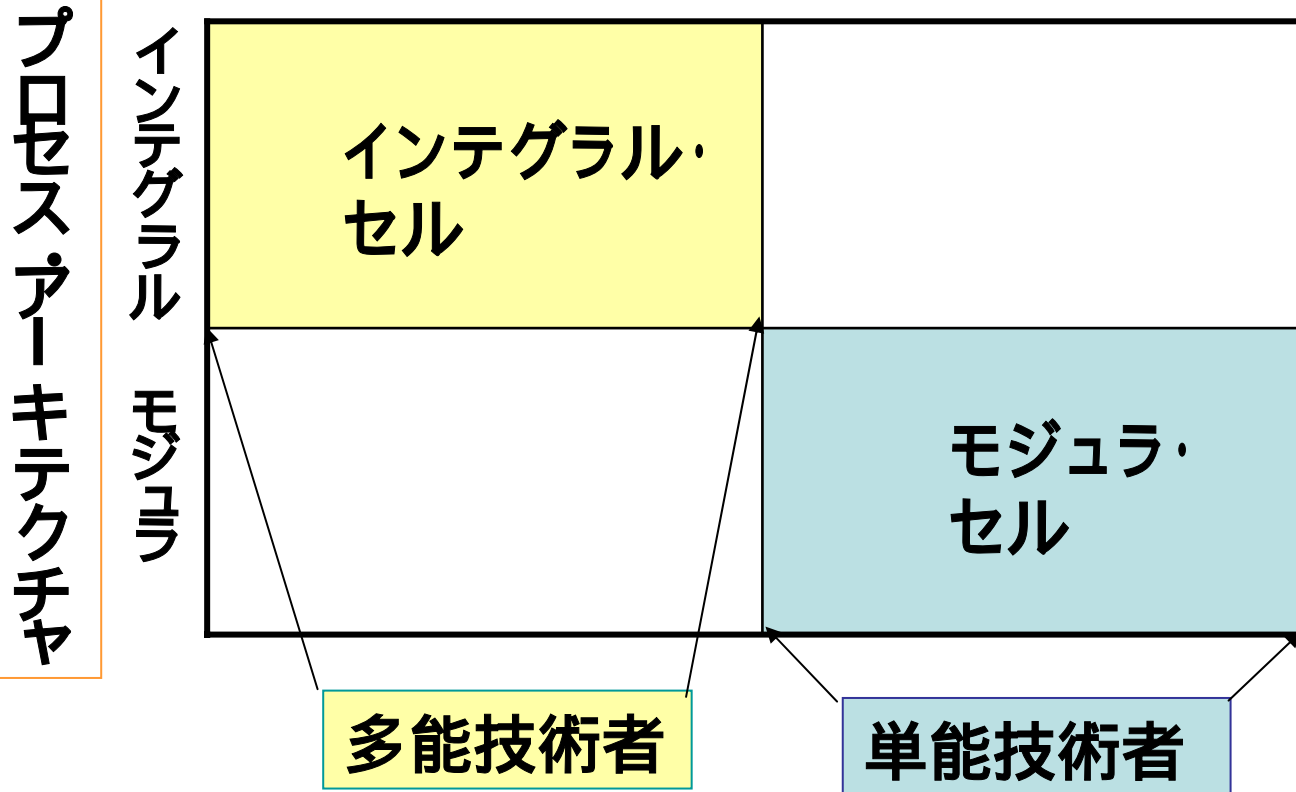
- 個々のプロダクトに関する事業を、「イノベーション・モード」と「コモディティ・モード」に分けている。
- イノベーション・モードでは、インテグラル・アーキテクチャを容認する。
- コモディティ・モードでは、デザイン・ルールを追加・挿入することによって、順次、モジュラ化を進める。
- しかし、モジュラ化することによってNOV (net option value)が上昇する場合は、強いてモジュラ化することはしない。
- コモディティ・モードでは、アーキテクチャ間の抽象レベル間で起きる循環性をすべて明らかにし、収束循環するものは許し、発散循環が存在する場合には、それが起きないように、デザイン・ルールを追加・挿入する。
- モジュラ化可能なデザインをPDTと称する「セル」、収束循環するデザインで、モジュラ化に適さないものをPITと称する「セル」に格付けする。
 - PDT: product development team (INCOSEに基づく)
 - PIT: product integration team (INCOSEに基づく)
- コモディティとしてのプロダクトに市場価値が失われてきたときには、できるだけ早く、再びインテグラル・モードへ回帰する。

プロダクト・アーキテクチャ、プロセス・アーキテクチャ、
そして「セル」：セルが、プロダクト、プロセスおよび技術者を組み合わせる。

プロダクト・アーキテクチャ

インテグラル

モジュラ



プロセス構造も、アーキテクチャによって可視化
できる。

プロセス・アーキテクチャ

セルとは

- セルは、単位プロダクトとそれを製作するためのマイクロプロセスを統合するためのカプセル 下記のを収容：
 - (定義されたスキルをもった)人に対するインタフェース
 - マイクロプロセス
 - セルで製作する単位プロダクト(アーティファクト)
 - ツール(DSL: domain specific languageおよびプログラムジェネレータ)
 - 手順指導書(ガイドライン)
- セル仕様
 - インタフェース(静的)
 - コントラクト(動的)

- 依存性
 - プロダクト依存性
 - 仕様(参照 – refers/referred、使用– uses/used)、品質および性能をパラメータとする。
 - プロセス依存性
 - 時間、役割、ツール環境をパラメータとする。
- 設計依存性 (design dependency)
 - 設計過程における、プロダクト依存性とプロセス依存性を統合した概念 (Baldwin and Clark)
- セルは、プロダクト依存性およびプロセス依存性を統合する必要から、設計依存性を基礎として組み立てる。

設計依存性の可視化

A、Bは、互いに依存性をもった単位設計ワークロード

	A	B
A		
B		

AおよびBは、互いに依存しない。

	A	B
A		
B	×	

Bは、Aに依存する。

	A	B
A		×
B	×	

AとBは、互いに依存する。

DSM (DesignまたはDependency Structure Matrix)



依存性グラフ
(dependency graph)

- AがB、またはBで製作するプロダクトを、参照 (refer)、使用 (use) している。
- Bの仕様、品質または性能が変更された場合、Aの正当性が失われる。
- AのプロセスがBのプロセスに依存する。

DSMを用いたアーキテクチャ変換

内部に2つのインテグラル・アーキテクチャを含むモジュラ・アーキテクチャへ変換したことの可視化

	1	2	3	4	5	6
1	-			×	×	
2		-			×	
3			-			
4				-		×
5	×				-	
6			×	×		-

変換



	3	6	4	1	5	2
3	-					
6	×	-	×			
4		×	-			
1			×	-	×	
5				×	-	
2					×	-

モジュラセルとインテグラルセルの識別

セル生産では、モジュラ・アーキテクチャでは禁じられているcannot-rule(赤印)を、インテグラル・アーキテクチャとして容認し、人智が支配するインテグラル・セルを行使して、循環依存を克服し、品質・生産性を向上する。

\$root		→	↷	↻	↺
schema	schema1 1	.	1		
	schema2 2	1	.		
	schema3 3			.	
	schema4 4	1	1		.

Lattix Inc.'s LDMを用いて、(有)アークウェイ・石黒尚子作成

とり上げる問題 (problem)

日本の組織的文化のなかから生まれたセル生産方式を、ソフトウェア生産に適用する方法を考えて見よう。

ソフトウェアファクトリ

基本概念

Greenfield, J., et al.,
Software Factories
(2004)

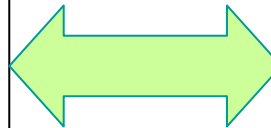
実践システム

再利用強化型ソフトウェア
工場(日本):1970年代に
実現

注



Microsoft's Visual Studio
Software Factories/
Team System



セルによる
ソフトウェア生産システム

注:筆者らは、1977年に2300人規模の再利用強化型ソフトウェア工場を発足させ、アプリケーションドメインを特定して、ソフトウェアアーキテクチャ・テンプレートおよびプログラムモジュールを系列化し、プログラムジェネレータを多用して、ソフトウェア生産の工業化を図った。

ソフトウェア工場組織

ソフトウェア工場長

総務・人事・労働・厚生

財務・会計

原価管理

生産管理・設備・資産

購買・外注

品質保証

技術管理

■ドメインA製品部

■シェフチーム

■コックチーム1

■コックチーム2

■ 以下、複数のコックチーム

■テストチーム

┆
┆ (途中省略)

■ドメインN製品部

■シェフチーム

■コックチーム1

■コックチーム2

■ 以下、複数のコックチーム

■テストチーム

プロダクトグリッド

		ソフトウェア開発視点			
		エンタプライズ	情報	アプリケーション	技術基盤
ソフトウェア開発レベル	概念レベル				
	論理レベル				
	物理レベル				

コマのなかには、プロダクトまたはアーティファクト

シェフモードで開発したプロダクトを記憶・管理するデータベース・スキーマ

ソフトウェア・プロダクトグリッド例

ソフトウェア開発視点

	エンタプライズ	情報	アプリケーション	技術基盤
概念レベル	1-1 システム仕様, システム目的, システム制約, 価 値・戦略分析, 品質 評価基準, 性能評価 基準	1-2 システムエンティティ,	1-3 システム/ソフトウェア/ヒュー マン・インタラクション, IEEE ConOps, システム (ビジネス)プロトタイピング, システムプロセス, 非機能 的要求, アスペクト要素と本 質要素の分離	1-4 設置計画, 環境制約,生存継続 性計画
論理レベル	2-1 用語定義, ソフトウェアアーキテ クチャ(論理的), コンポーネントおよび インターフェース/コン トラクト	2-2 情報フローモデル, データ定義,CRUD	2-3 I/Oリスト, センサ仕様, アクチュエータ仕様, サービス(制御,計算)定義, サービスアルゴリズム, イベ ント/アクティビティモデル,	2-4 フレームワーク定義, 基盤定義, 配置計画, ネットワーク設計仕様
物理レベル	3-1 ソフトウェアアーキテ クチャ(物理的), コンポーネント定義 (プログラム・コンポー ネント)、プロジェクト セル構成	3-2 データベース定義,	3-3 GUIデザイン, プレゼンテーションモデル, トランザクションモデル,コン ポーネント・コラボレーション, イベント定義, アクティビティ定義	3-4 配置実装,保守計画 ConOps: Concept of Operations

**Baldwin and Clarkのデザイン・パラメータに従って、
プロダクトを5カテゴリに分ける。**

- **Categories of Design Parameters**
 - **External Parameters**
 - たとえば、プロジェクト憲章、要求仕様書、ユースケース(本質コンサーンと側面コンサーンの分離)、用語定義
 - **Design Rules**
 - 以下の3項目をモジュラ化するために守らねばならないルール、セキュリティ、品質および性能保証計画
 - **Application (functional) Modules**
 - たとえば、業情報モデル、データベース・スキーマ、ビジネス・イベントリスト、ビジネス構造モデル、
 - **Subsidiary Modules**
 - たとえば、ビジネスプロセス / アクティビティモデル、CRUD
 - **Application Controller**
 - ユーザインタフェース / ユーザプロセス・コントロール、外界同定 / ビジネスプロセス・コントロール

Baldwin, C.Y., and K.B. Clark, Design Rules, Vol.1, The MIT Press (2000)

デザイン・ルール

デザインアーキテクチャをモジュラ化するために、重要な役割を果たす基準(たとえば、下記)を定めたもの。

- **コンポーネント基準**
 - **User profiling**
 - User management
 - **Access control**
 - security
 - **Data integration**
 - Organizational management, entity/relationship
 - **Work flow**
 - State management, process/activity management
 - **Event notification**
 - Event management, notification management, messaging management
- **ユーザインタフェース基準**
 - Data integration service, access control service, user profiling service, work flow service, event notification service, trading service
- **エンタプライズ・ソリューションパターン**
 - Webプレゼンテーションパターン、配置・実装パターン、サービスパターン、
 - パフォーマンス・信頼性パターン

モジュラ化されたデザイナーキテクチャ

青色の表示は、循環依存の残存を教える。DR、サブAMを新しく追加するなどしてモジュラアーキテクチャに変換する方法(注)と、インテグラル・セルで人智的に克服する方法がある。

External Parameters

Design Rules

Application Modules

Application Controller

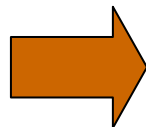
		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	
E P	1	*				X														
	2	X	*																	
	3		X	*	X															
	4		X		*															
	5	X			X	*														
D R	6	X		X			*													
	7			X				*												
A M	8						X		*		X		X							
	9							X		*										X
	A							X		X	*									
	B								X	X		*								X
	C											X	*							
A C	D													*		X				X
	E	X		X							X			X	*					
	F				X						X			X		*			X	
	G											X				X	*			
	H		X															X	*	
	I												X	X					X	*

プロダクトグリッドをモジュラ・デザイナーキテクチャへ変換

シェフモード

コックモード

	エン タ プ ラ イ ズ	情 報	ア プ リ ケ ー シ ヨ ン	技 術 基 盤
概 念 レ ベ ル				
論 理 レ ベ ル				
物 理 レ ベ ル				



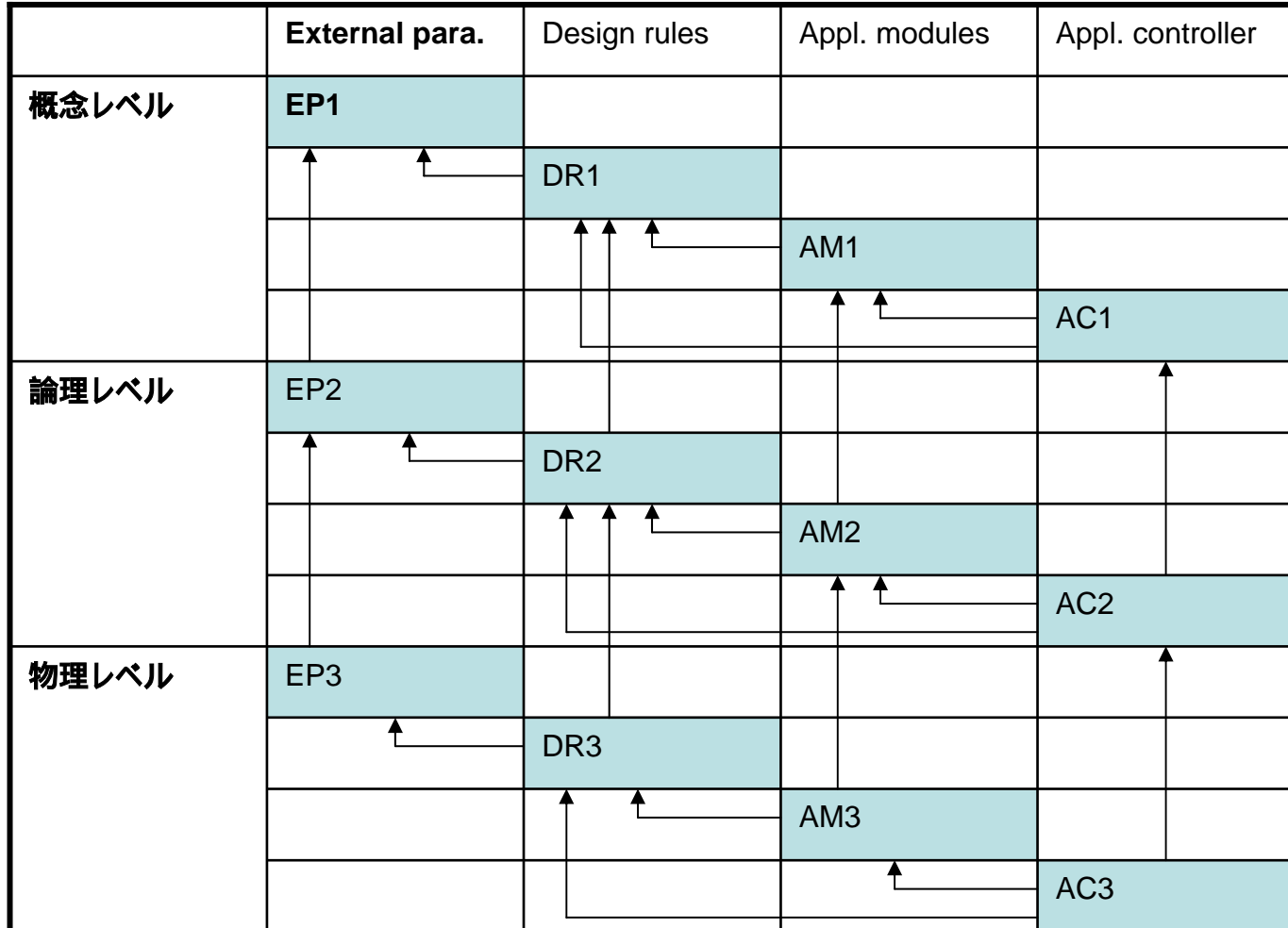
概 念 レ ベ ル	E P																			
	D R																			
	A M																			
	A C																			
論 理 レ ベ ル	E P																			
	D R																			
	A M																			
	A C																			
物 理 レ ベ ル	E P																			
	D R																			
	A M																			
	A C																			

このセクションが空白の場合に、モジュラアーキテクチャ

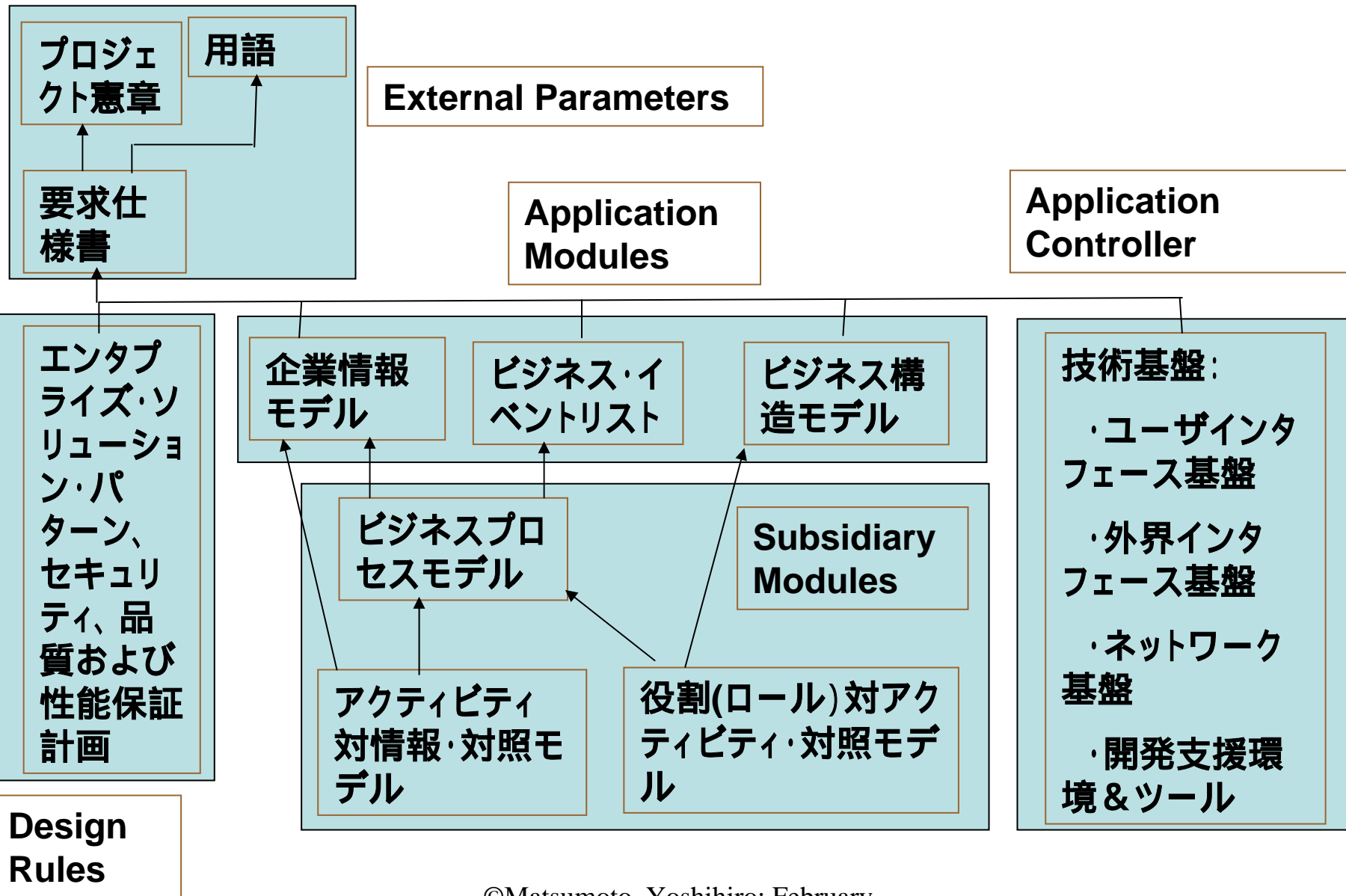
EP: external parameters, DR: design rules, AM: Application modules, AC: application controller

理想とするモジュラ・ソフトウェア・デザイナーアーキテクチャ

矢印の意味: a → b aは、bに依存する。

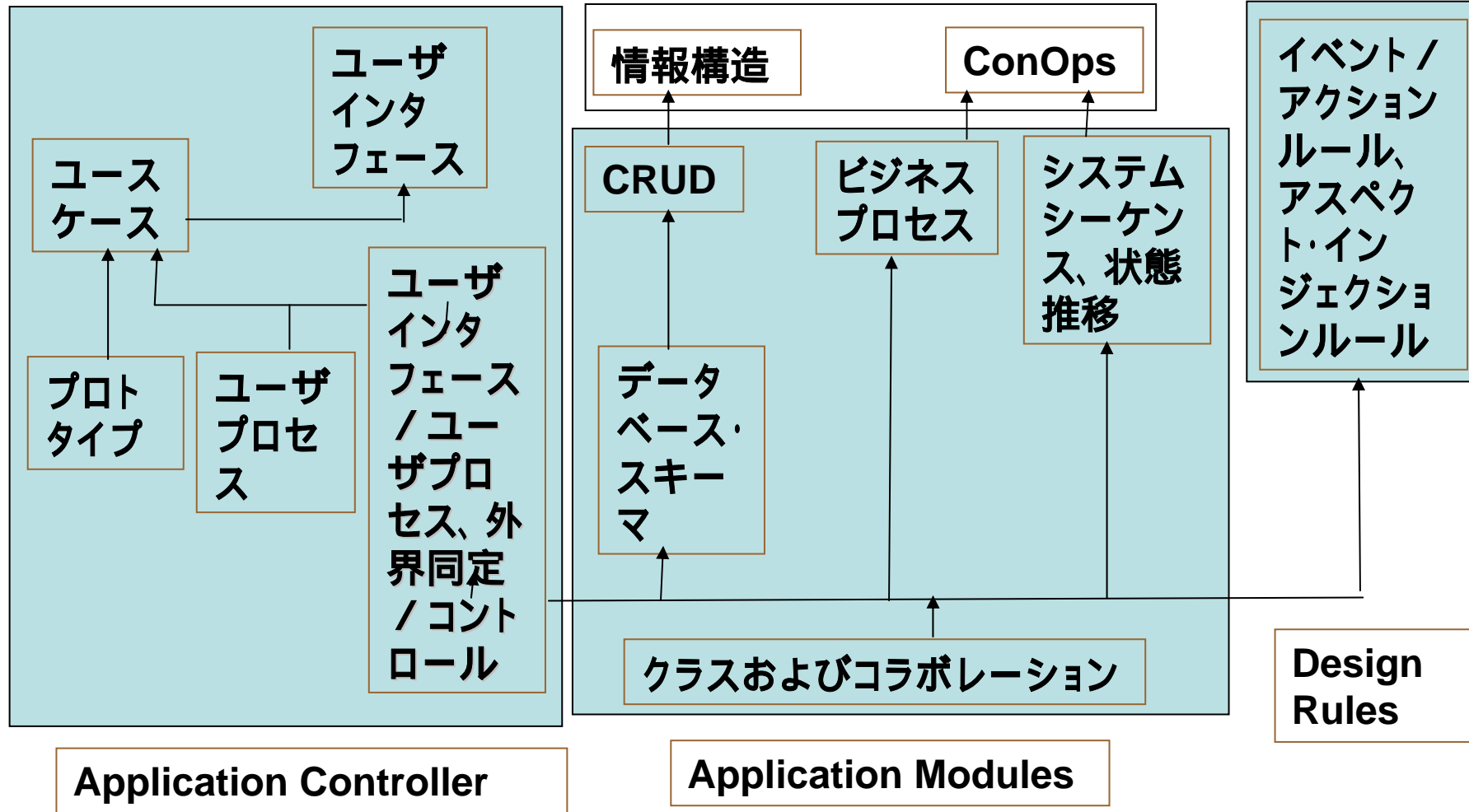


概念レベルのデザイン依存性グラフ例



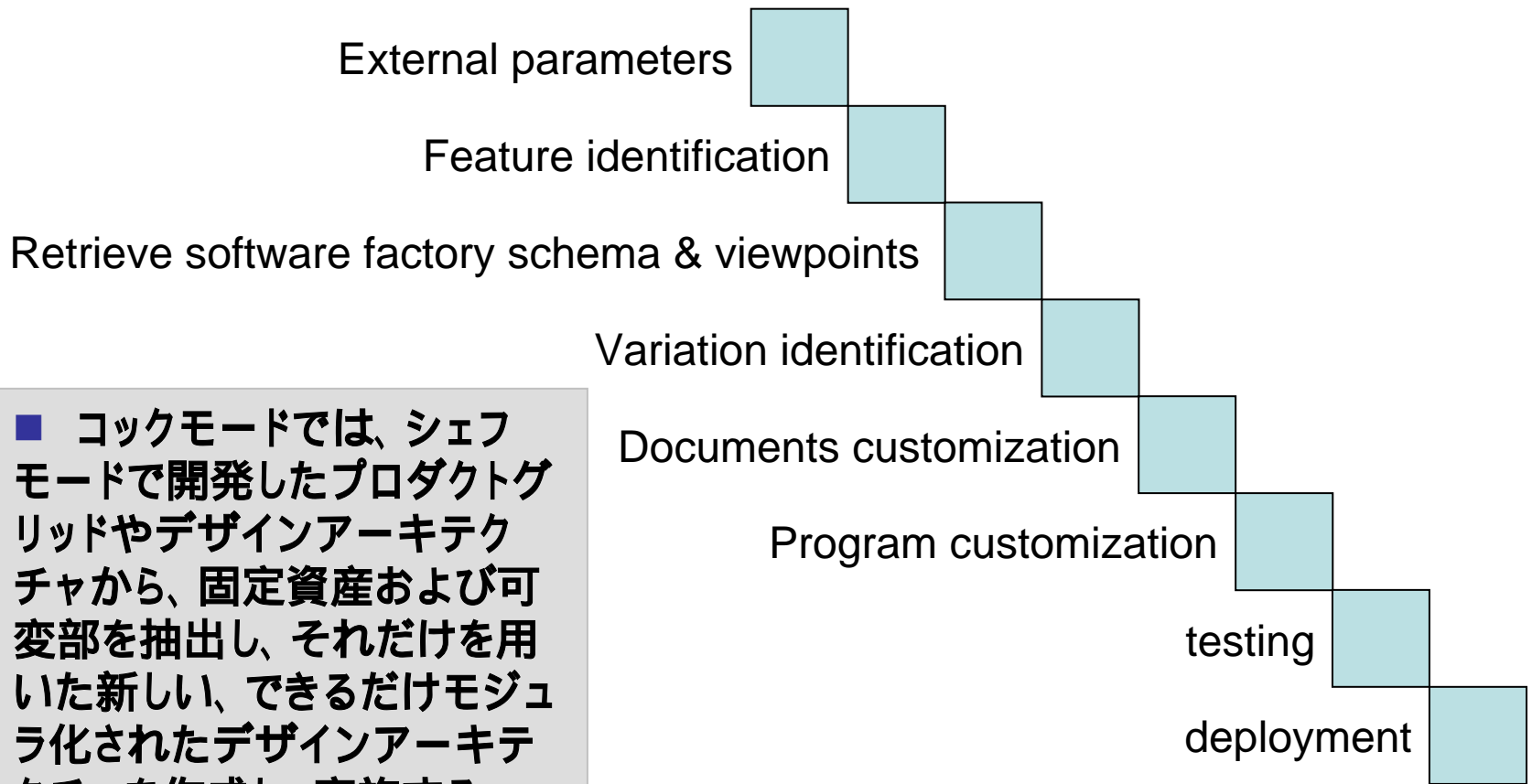
論理レベルのデザイン依存性グラフ例

概念レベル全体をexternal entityとみなし、論理レベルのデザイン全部を依存させる。



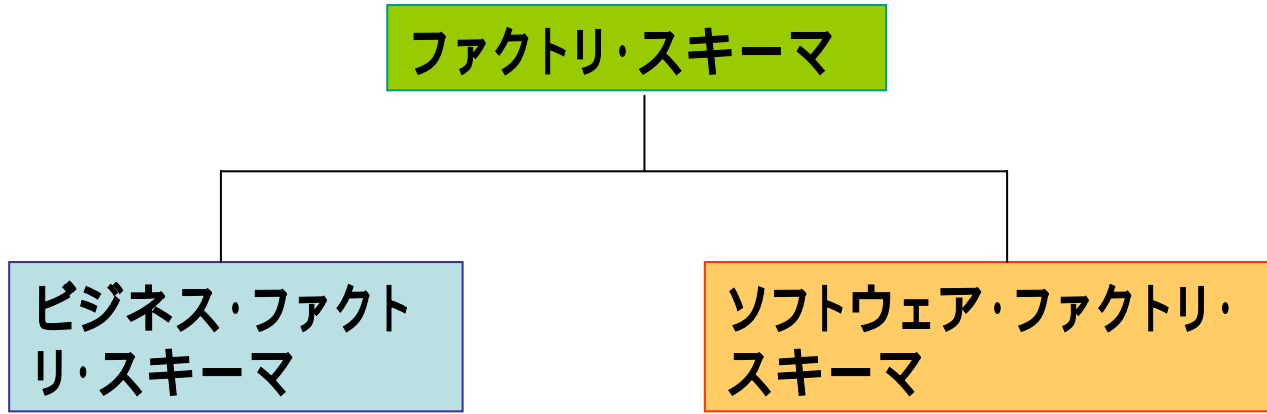
コックモードのデザイナーアーキテクチャを作成する

■ シェフモードからコックモードへの変換チーム：これがソフトウェアファクトリの心臓部



■ コックモードでは、シェフモードで開発したプロダクトグリッドやデザイナーアーキテクチャから、固定資産および可変部を抽出し、それだけを用いた新しい、できるだけモジュラ化されたデザイナーアーキテクチャを作成し、実施する。

ファクトリスキーマ



ソフトウェア・ファクトリ・スキーマ

ソフトウェア・ファクトリ・
スキーマ

ソフトウェア・セル

マッピング・アロー
(矢)

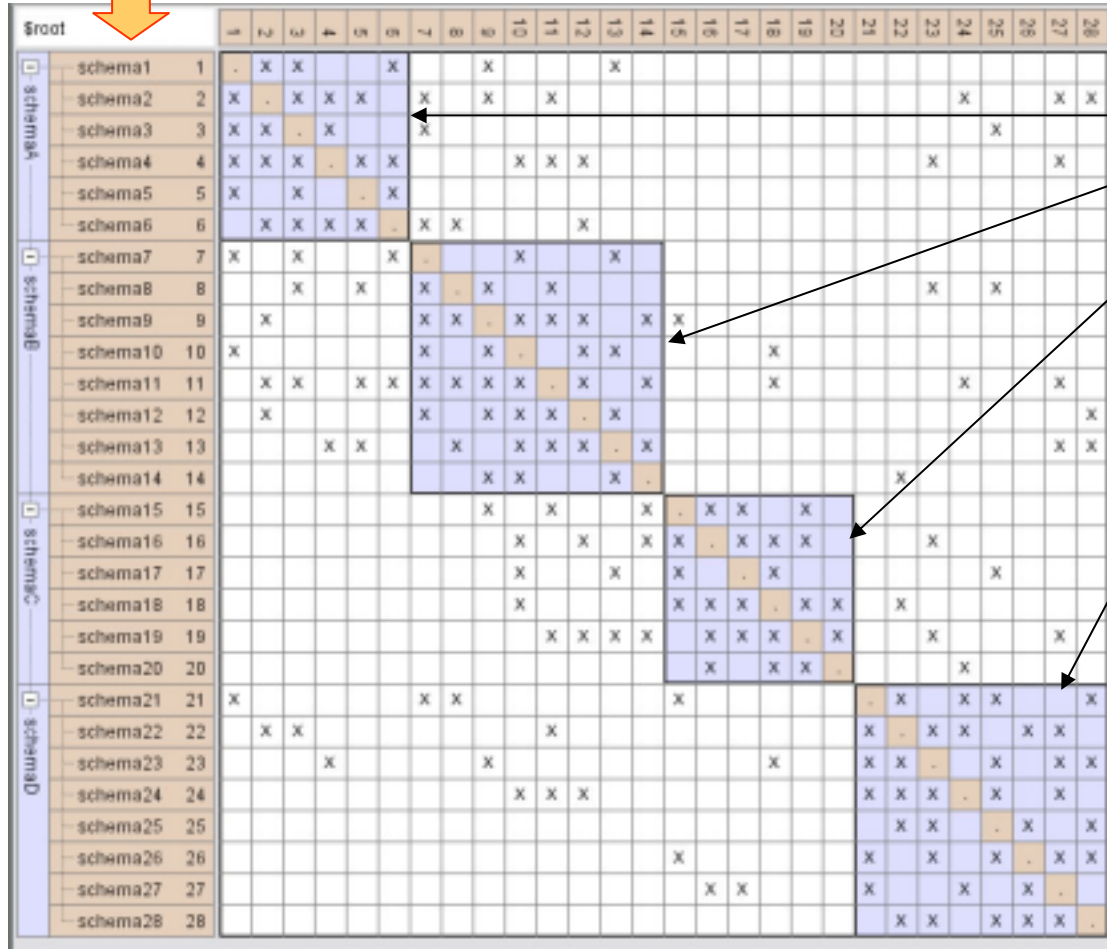
ビューポイント

マッピング・アロー
(矢)

DSMとの関連

ビューポイント

×印のあるところは、マッピング・アローで表現する。

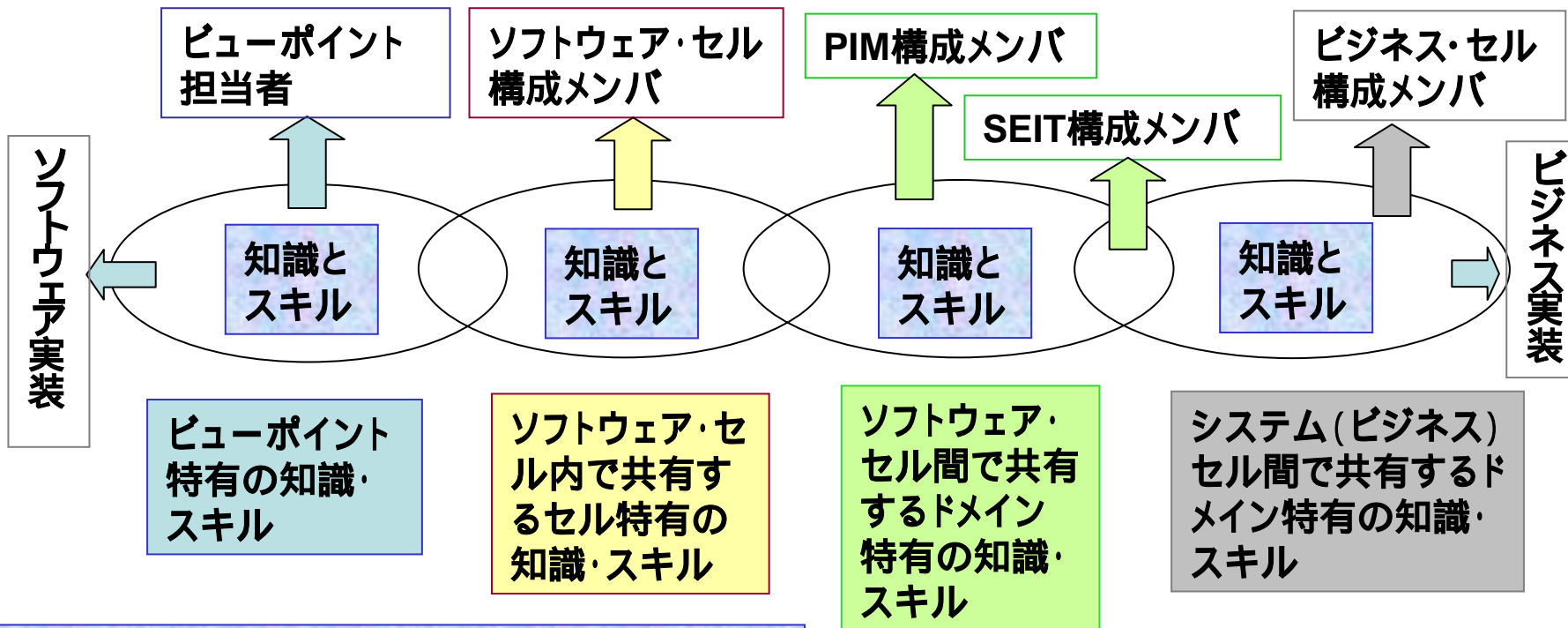


ソフトウェア・セル

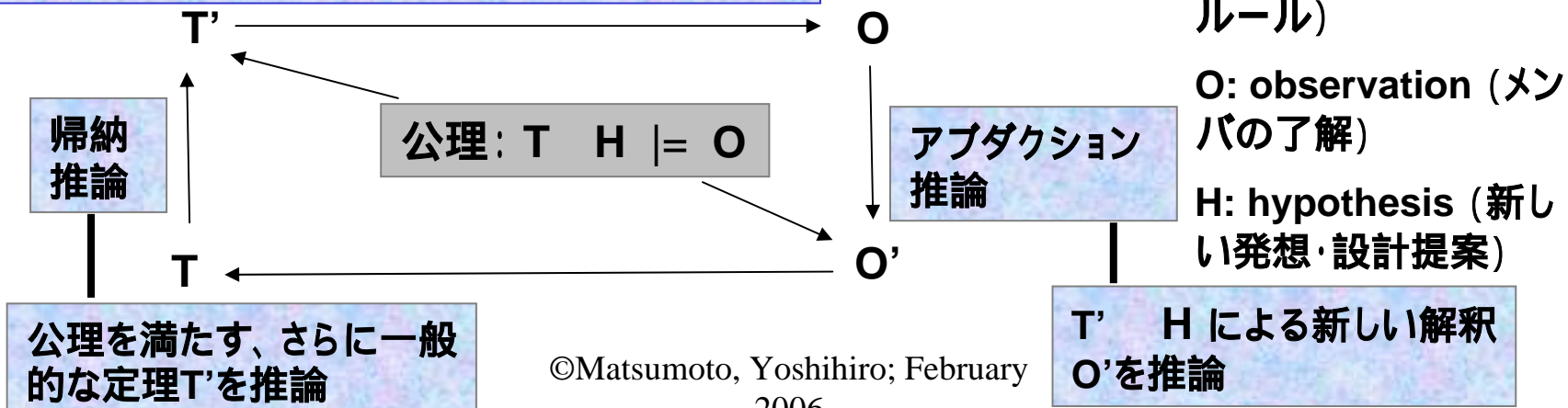
INCOSEへの対応

- セル内で逆戻り依存がある場合は、セル内のコミュニケーションで解決する。セルは、PDT (Product Development Team)に相当。
- 異なるセル相互間に、逆戻り依存がある場合は、依存サイクルを明確にし、発散しないようにデザインルールを追加し、PIT (Product Integration Team)を組織して、依存によって派生する問題を解決する。
- ビジネス・ファクトリ・スキーマ内セルとソフトウェア・ファクトリ・スキーマ内セルの間に、逆戻り依存がある場合には、SEIT (System Engineering Integration Team)を組織して、依存によって派生する問題を解決する。

ファクトリ・セルのなかでのコミュニケーション



セル・コミュニケーションにおける設計思考



セル編成上の留意点

- モジュラ・セル(INCOSEの呼称ではPDT)が望ましい。
 - セル内部に循環コミュニケーションがあっても、セル内部で収束するようなセルである。
 - 全てのセルが上記の条件を満たし、セル相互の間では、モジュラ・アーキテクチャ (Baldwin & Clark) を形成する。
- ひとつのセルから他のセルへ向う循環コミュニケーションが存在し、モジュラ・セルおよびモジュラ・アーキテクチャを実現することが困難であるか、またはそのために投資しなければならない NOV (net option value) が大きくなるような場合は、つぎのように考える。
 - 循環コミュニケーションが存在する複数のセル間において、循環コミュニケーションが必ず収束するように、デザインパラメータを修正する。
 - その上で、当該複数セル間で、依存パスを明確にした上で、インテグラル・セルを編成する。
 - このインテグラル・セルは、INCOSEの呼称では、PIT(複数のPDTのまたがる場合)、またはSEIT(システム(ビジネス)領域にまでまたがる場合)に相当する。

全レベルを通じたデザイン・アーキテクチャから、セルを編成する

モジュラ化ができない項目を識別して、インテグラル・セルを組織する。

	エンタプライズ	情報	アプリケーション	技術基盤
ビジネス戦略				
ビジネス戦術				
ビジネス執行				
ソフトウェア開発 概念レベル				
ソフトウェア開発 論理レベル				
ソフトウェア開発 物理レベル				



垂直連携セル



水平連携セル

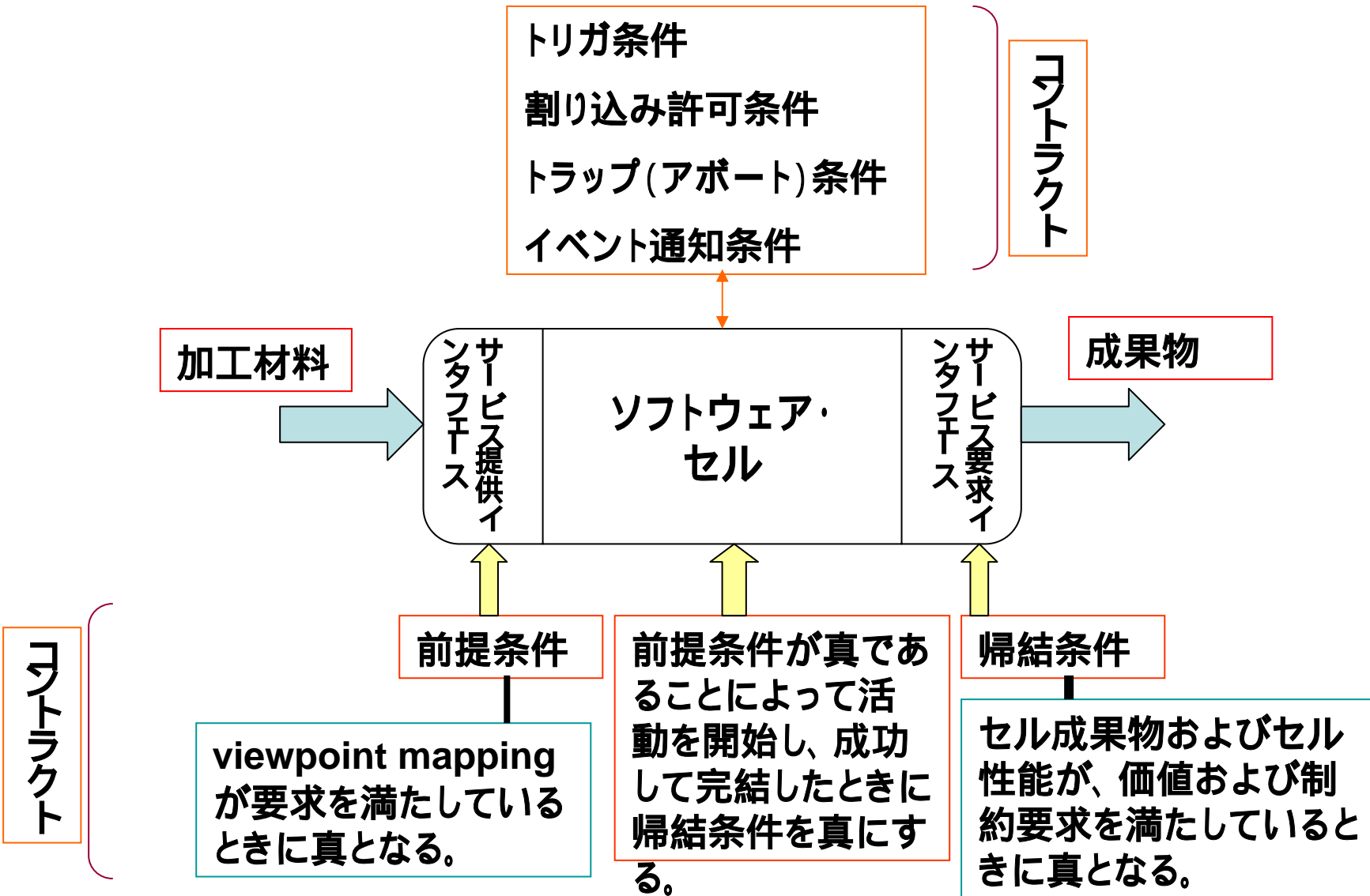


複合連携セル

セルにおけるユニット・ワークロード定義表

UW No.	Name of Responsible Person	
UW Name	Purpose of UW	
Precondition	Background (context)	
UW Denotation	Objectives	
Import documents	Estimated artifacts size:	
Export documents	Expected quality level:	
Process denotation	Expected cost:	
	Expected productivity:	
	Expected reuse rate:	
	Constraints	
	Deployment conditions:	
	Performance:	
	Schedule:	
Post-condition	Standards, methodology, tool	

ソフトウェア・セルのインタフェースとコントラクト



セルは、INCOSE/SE Handbookに準拠

- 「セル」は、INCOSE/SE Handbookに準拠して説明できる。
 - Product Team
 - PDT: Product Development Team
 - 「セル」
 - PIT: Product Integration Team
 - セルのなかの代表者によって構成する「インテグラル・セル」
 - SEIT: System Engineering and Integration Team
 - PITのなかの代表者によって構成する「インテグラル・セル」



SYSTEMS ENGINEERING HANDBOOK

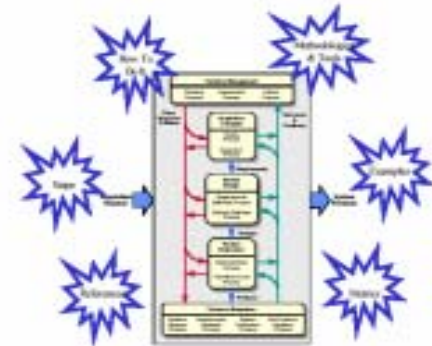
A "WHAT TO" GUIDE FOR ALL SE PRACTITIONERS

INCOSE-TP-2003-016-02, Version 2a, 1 June 2004

Released by:

Technical Board

International Council on Systems Engineering (INCOSE)

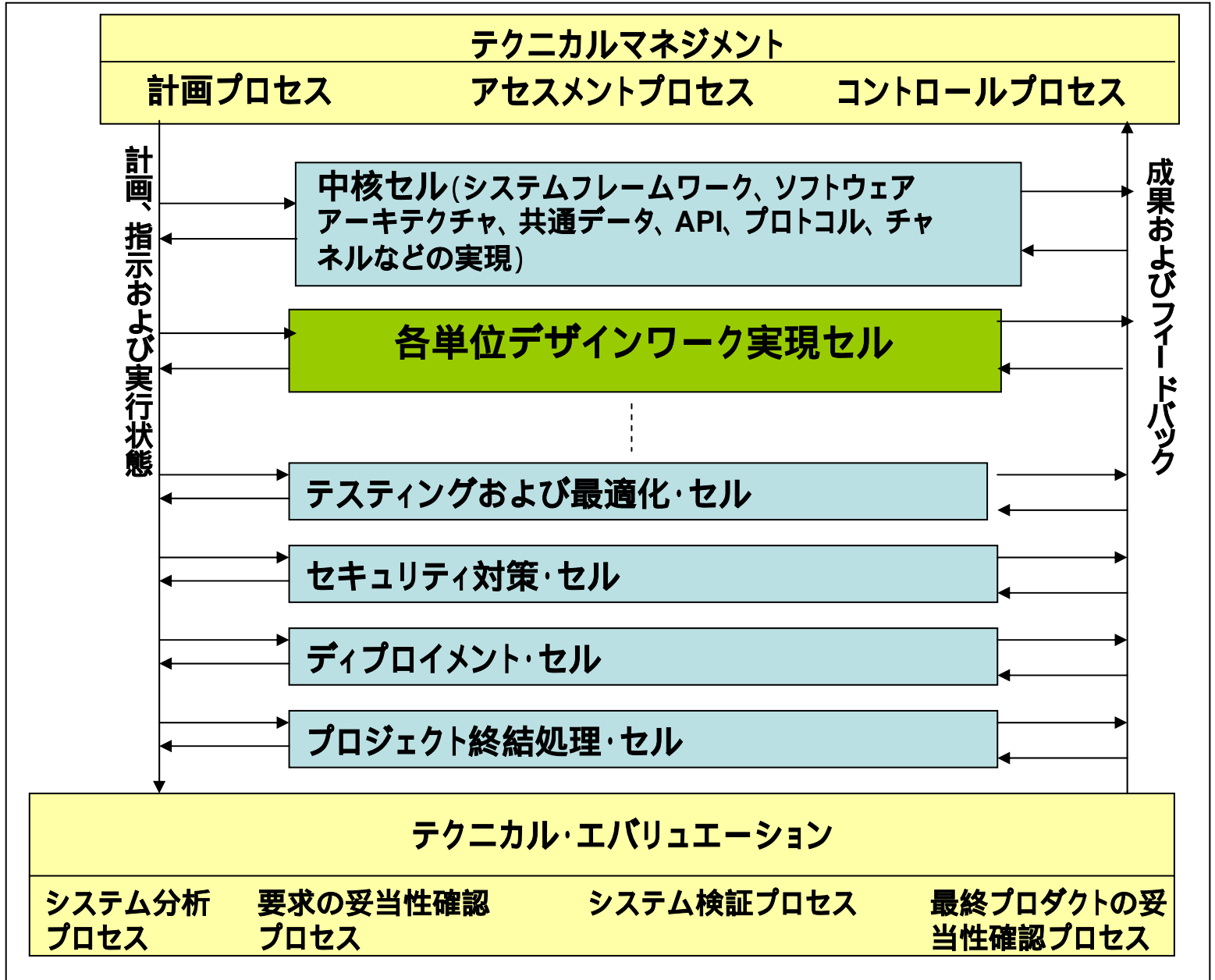


Copyright (c) 2002, 2004 by INCOSE, subject to the restrictions on the following page.

SEIT

PIIT

PDT



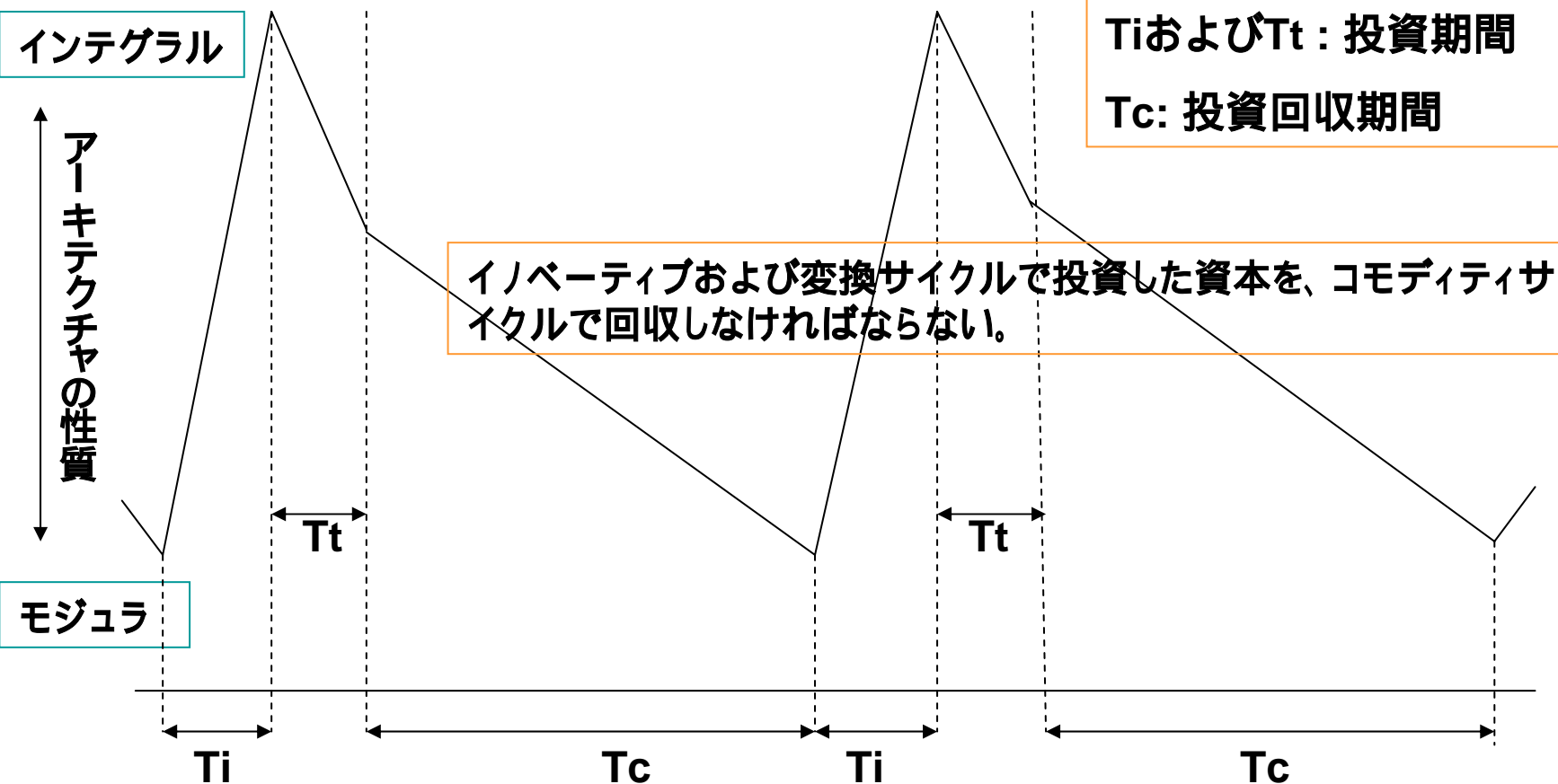
ビジネスエコシステムに即応できるようにセルは並行繰返し実行

解 (Solution)

ソフトウェア・セル生産方式の実践方法を考えてみよう。

イノベータイプ(シェフ)・サイクルとコモディティ(コック)・サイクル

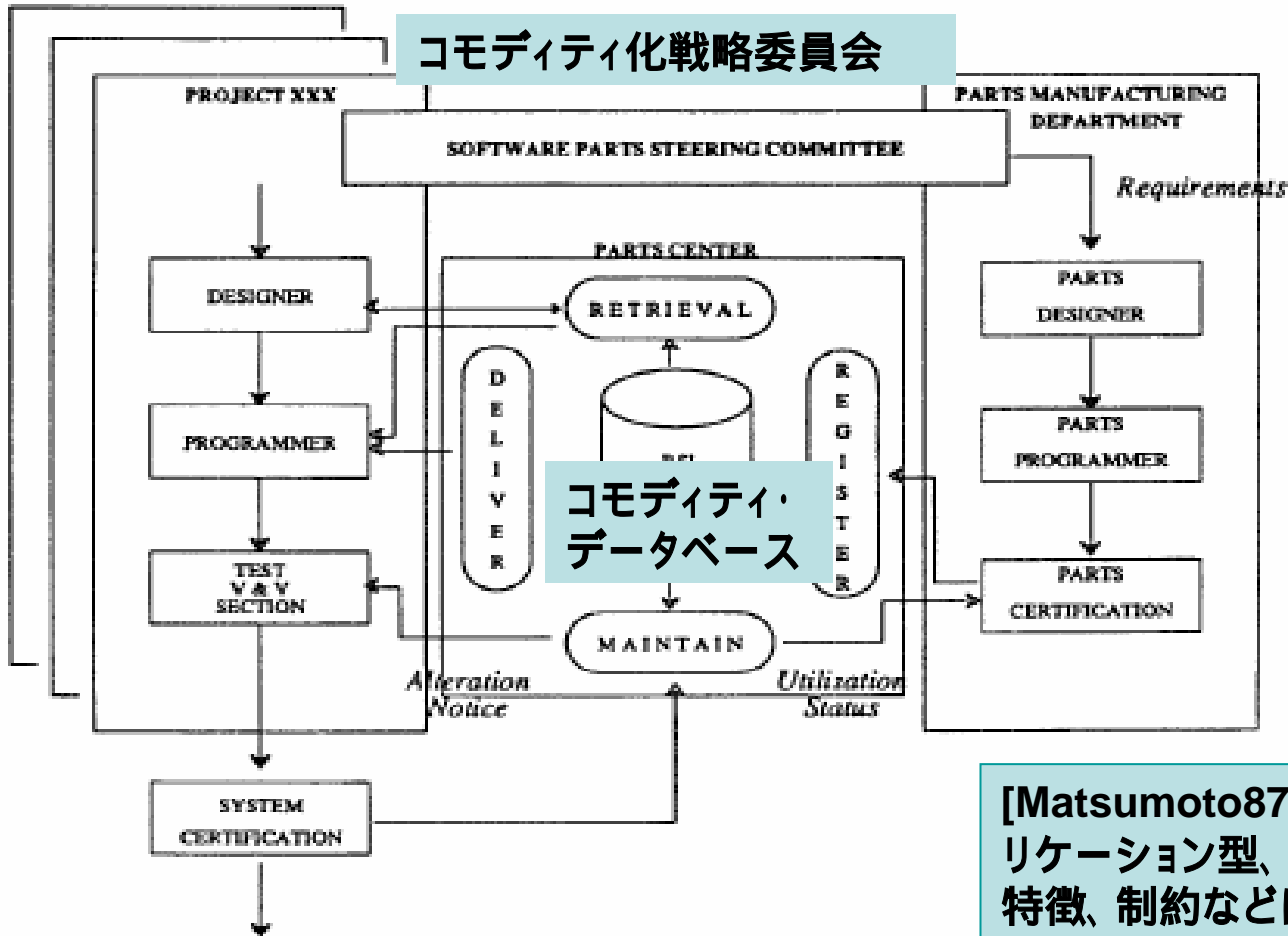
- T_i : イノベータイプ・サイクル、 T_c : コモディティ・サイクル
- T_t : インテグラル・アーキテクチャからモジュラ・アーキテクチャへの変換サイクル
- T_i+T_t : **開発リードタイム (企業独自の深層パフォーマンスで短縮)**



インテグラル・アーキテクチャからモジュラ・アーキテクチャへの変換機構 (再利用率部品を生産し、管理し、サービスする組織)

コモディティプロシエクト

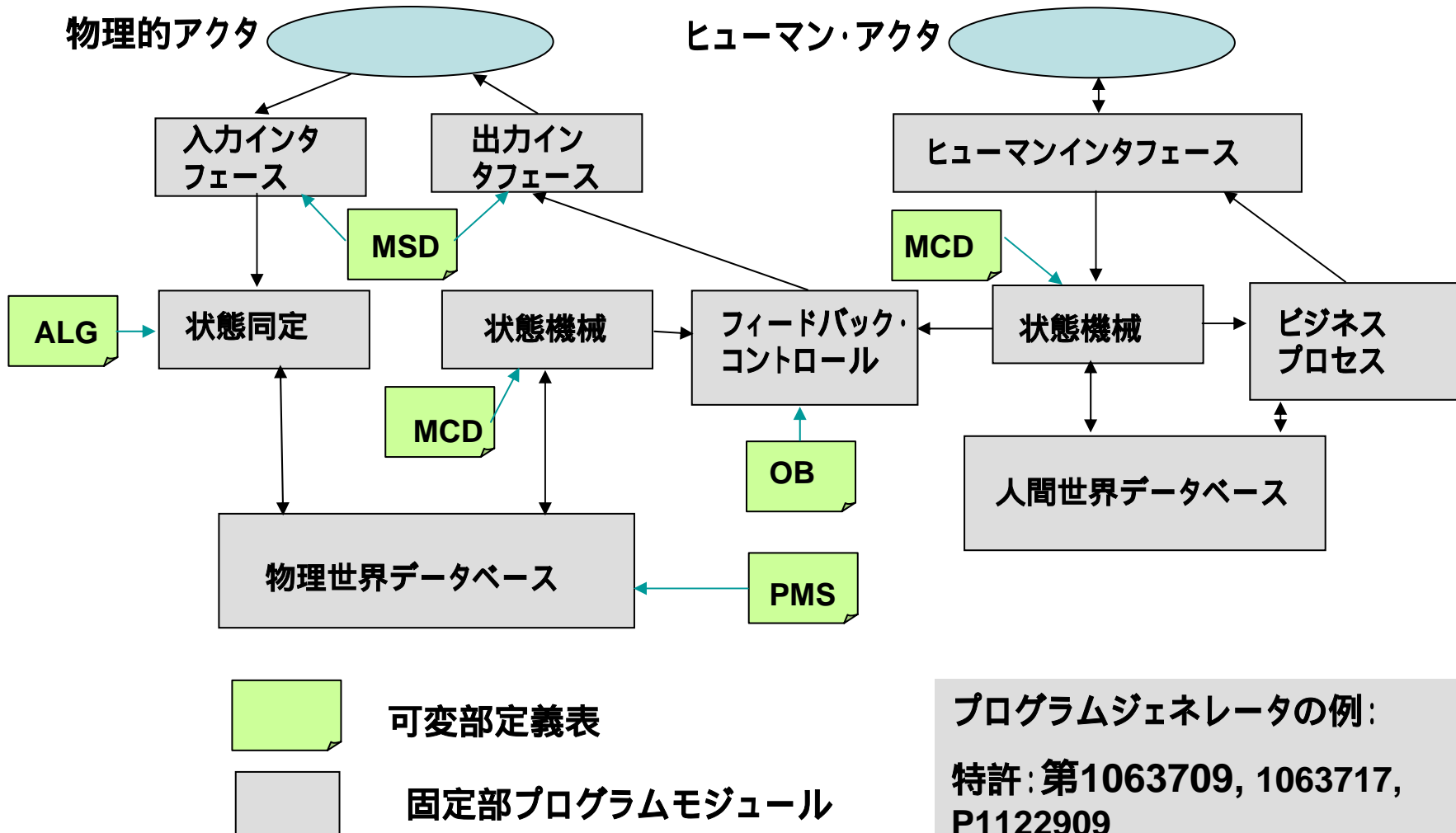
イノベーションプロシエクトまたは



シエラ コックアーキテクチャ変換チーム
コモディティ開発常設チーム

[Matsumoto87]より引用: アプリケーション型、部品パターン、特徴、制約などによって、部品を検索できるようにする。

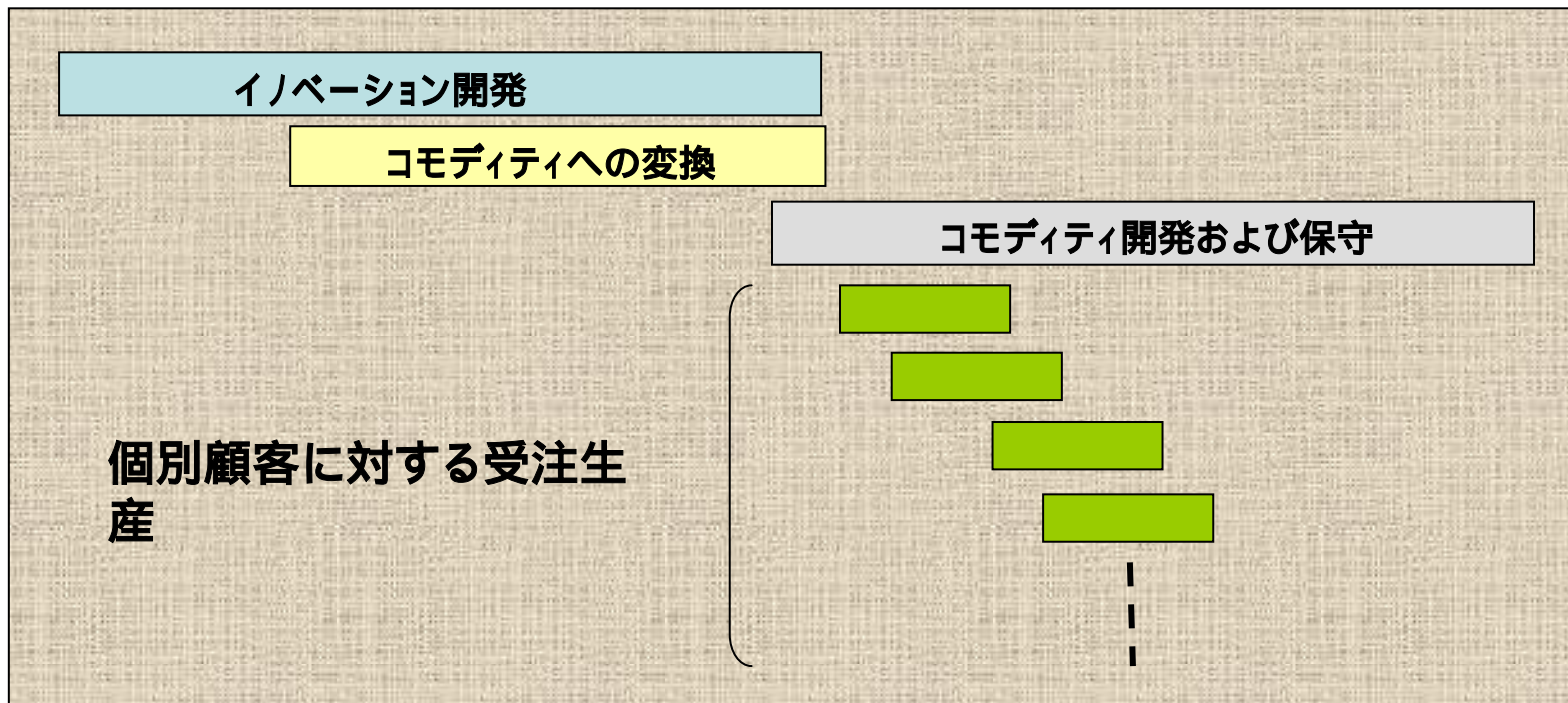
シェフモードからコックモードへの変換： 固定部と可変部の分離およびプログラムジェネレータ例



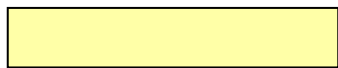
プログラムジェネレータの例：
特許：第1063709, 1063717, P1122909

ソフトウェアファクトリにおけるマネジメントサイクル

あるひとつの製品系列の開発および顧客への提供



イノベーション・ステージ(シェフモード)



コモディティ化のための変換ステージ(シェフ コック変換)



コモディティ開発および保守ステージ



顧客からの注文に応える受注生産ステージ

ソフトウェア・セル生産方式の成功要件

- コンピテンスをもった経営者・管理者
- 先見性をもった投資計画、投資回収計画
 - シェフモードでの開発投資
 - コックモードへの変換投資
 - 再利用資産の維持・改善・更改投資
- 技術者の意欲と熱意
- エンジニアリング・エコノミクス能力開発
 - NOP (net option value: Baldwin and Clark)
 - NPV (net present value), IRR (internal rate of return)
- デザインルール、可変要素マネジメント
 - コンポーネント、アスペクト、XML (XPathなど)、dependency injection

ソフトウェア・セル生産のキーポイント

- 事業対象となるアプリケーションドメインを、限定し、明示すること。
- シェフモードとコックモードを峻別すること。シェフモードでの開発方法論は、セル生産・議論の対象外。
- シェフモード開発は、自己資金で行う。シェフモード開発・初製品は、納入後、高品質であること。投資回収は、コックモードで実現する。
- 初製品納入後、アーティファクト(半製品および製品)を、プロダクトグリッド(3×4)に整理し、記憶・管理すること。
- プロダクトグリッドを、デザイナーアーキテクチャ(DSMを用いて表示: external parameters, design rules, application modules, application controllerから構成)に変換し、モジュラ化すること。デザイナーアーキテクチャのなかに、アジャイル部分(アスペクト、injected dependency、XML化された可変データ)を織り込む。
- デザイナーアーキテクチャのなかで、モジュラ・部分アーキテクチャとインテグラル・部分アーキテクチャを識別し、前者をモジュラ・セル、後者をインテグラルセルに対応させ、セルを編成する。
- セルの組織化およびマネジメントは、INCOSE Handbookに従って実施する。

むすび

- **ビジネスイノベーションやソフトウェア開発は、ガーデニング(庭作り: gardening)のようなもので、基本は恒に変わらない。**
- **ガーデニングの基本が自然と一体になることであるように、ビジネスやドメインを支配する原理・原則が、時代によって変わることはない。恒にアプリケーション・ドメインと一体になることが必要である。時代とともに変わるのは、市場感覚、組織、基盤およびツールのような人工物だけである。**
- **日本の産業組織文化を直視し、身近な先輩から受ける教訓や経験的指導を重視しなければならない。**

引用文献

- **Matsumoto's original papers:**
- **[Matsumoto87] Matsumoto, Y.: A software factory: An overall approach to software production, in "Software Reusability" ed. by P. Freeman, pp.155- 178, IEEE Computer Society(March 1987)**
- **[Matsumoto92] Matsumoto, Y.: Toshiba Software Factory, in "Modern Software Engineering, P.A.Ng and R.T.Yeh (eds.), pp.479-501, Van Nostrand Reinhold, 1990 Matsumoto, Y: Japanese Software Factory, Advances in Software Science and Technology, Vol.4, Japan Society for Software Science and Tehnology, pp.21-42, Iwanami Shoten, Tokyo(1992)**
- **[Matsumoto93] Matsumoto, Y.: Japanese Software Factory, in "The Encyclopedia of Software Engineering", J.J.Marciniak (ed.), pp.593-605, John Wiley & Sons, New York, 1993**

- **The papers which refer the Matsumoto's papers:**
- **[Basili92] Basili, V.R., and G. Cantone: A Reference Architecture for the Component Factory, ACM Transactions on Software Engineering and Methodology, Vol 1, No 1, January 1992, Pages 53-80**
- **[Blum92] Blum, B.: Software Engineering: A Holistic View, Oxford University Press (1992)**
- **[Cusumano91] Cusumano, M.A.: Japan's Software Factories, Oxford University Press, New York (1991)**
- **[Endres03] Endres, A., and D. Rombach: A Handook of Software and Systems, Addison-Wesley (2003)**
- **[Sage90] Sage A.P. and J.D. Palmer: Software Systems Engineering, John Wiley & Sons (1990)**
- **[Schach93] Schach, S. R.: Software Engineering, 2nd Edition, McGraw-Hill (1993)**
- **[Sommerville95] Sommerville, L.: Software Engineering, 5th Edition, Addison-Wesley (1995)**
- **[Vilet93] Van Vilet, H. and V. Van Vilet: Software Engineering: Principles and Practice, John Wiley & Sons (1993)**