MITSUBISHI SPACE SOFTWARE

大規模組込システムの要求分析、システム方式設計、そして、ソフトウェア設計までをつなぐモデルベース設計手法

関西事業部　藤原

三菱スペース・ソフトウエア株式会社

経験発表の内容

- はじめに
  - ✓ 弊社の事業分野
  - ✓ 昨今の開発形態（スクラッチ／流用／派生）
- 手法の概要
- システム要求分析
- システム方式設計
  - ✓ ドメイン分割
  - ✓ 方式設計の流れ
  - ✓ 機能方式設計
  - ✓ 基盤方式設計
- ソフトウェア要求分析、方式設計
- 手法の適用と定着に苦労したところ
- さらなる改善 – 機能安全への拡張
- さいごに

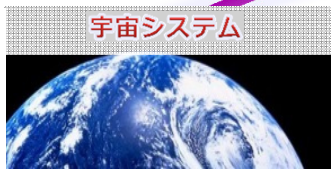
＜はじめに＞

本手法を適用している事業の説明

弊社の事業分野

さまざまな事業分野の多岐に亘る、ソフトウェア開発及び、システム設計支援・開発を請負にて行う会社。

ロケット航法・誘導解析(JAXA)・
人工衛星搭載ソフトウェア(JAXA)・
衛星管制システム・
衛星通信システム・



・高速通信ネットワークシステム
・特殊用途通信システム
・社会インフラシステム
・ネットワークセキュリティ(製品)

・カーエレクトロニクス
・カーマルチメディア（ナビゲーション）
・電動パワーステアリング
・EVインバータ制御

三菱リージョナルジェット・

・DNAバンクシステム
・ゲノム解析システム
・解析ソフトウェア

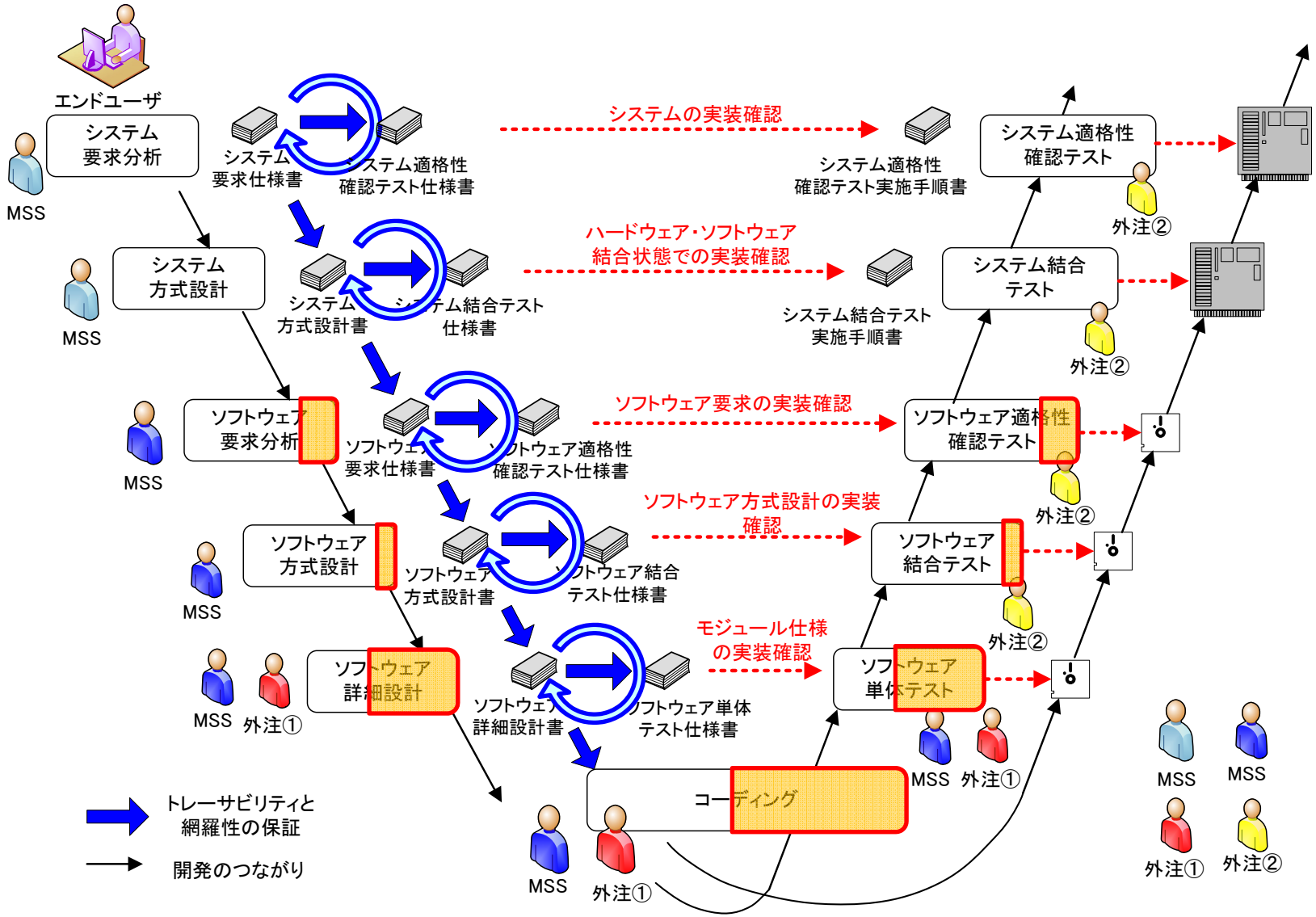
新幹線地震防災システム・
自治体向け防災システム・
海底地震津波システム・
緊急地震速報システム(ASP製品)・

・研究機関向け基幹システム
・データ解析システム(官公庁)
・企業向け基幹システム

・粒子線治療計画装置
・胸部X線診断支援システム（製品）

昨今の開発形態

システムの開発とテストのWモデルを下図に示す。設計と対になるテスト仕様書作成は、設計の間違いやモレを発見し易くするために、原則、同一者が役割を担う。テスト項目の十分性はレビューで確保する。



手法の概要

モデルベース・システムズ・エンジニアリング（MBSE)の流れ

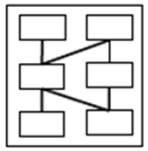
MITSUBISHI SPACE SOFTWARE

本手法は「システム要求分析～方式設計まで」を下記の流れ（MBSE)で設計する。

システム要求分析

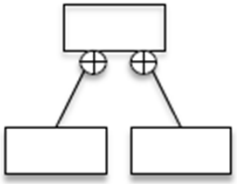
システム方式設計

①開発対象内で扱うドメイン用語の構造定義



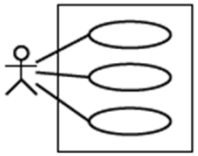
オントロジーモデルなど

②システム要求の構造化



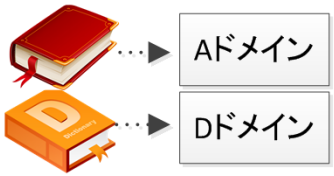
要求図

③ユースケースとシステム境界の決定

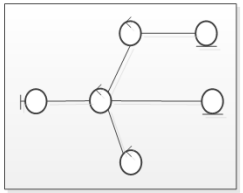


ユースケースモデル
／システムコンテキスト図

④ドメイン分割

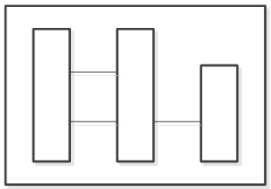


⑤ユースケースの振る舞い定義

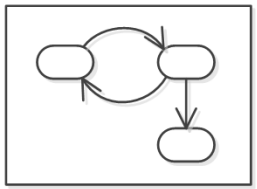


ロバストネス分析図
／シーケンス図

⑥システムを論理的に機能分割する

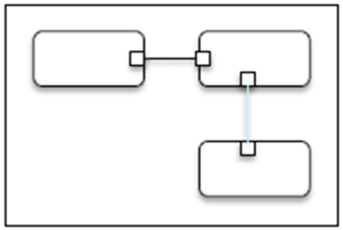


ブロック定義図

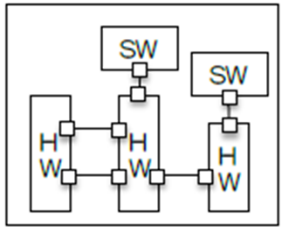


状態遷移図

⑦システム制約を検討し、HWとSWに分割する



パラメトリック図



内部ブロック図

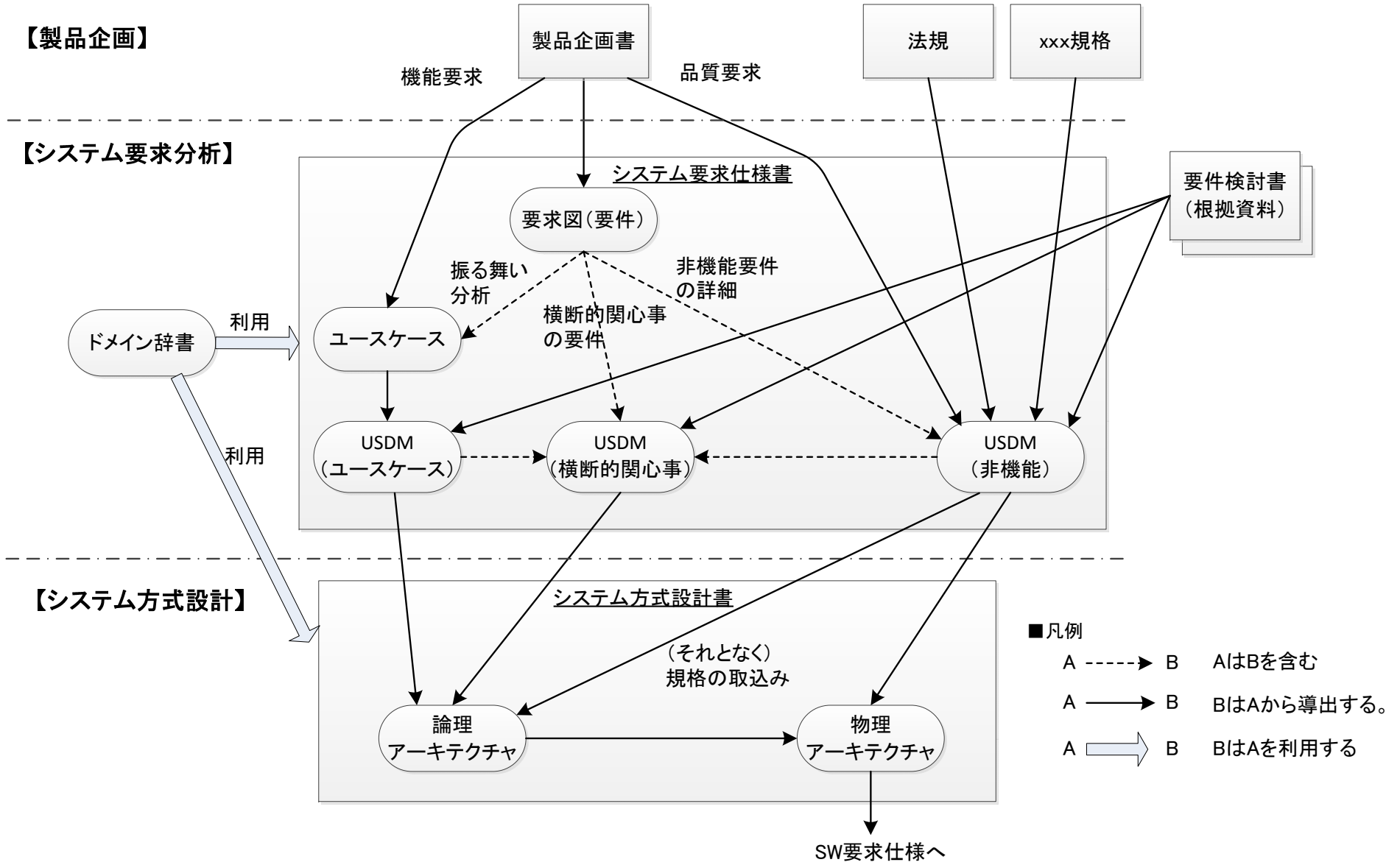
論理アーキテクチャ

物理アーキテクチャ

【凡例】

→ 設計分割境界に関して、矢印元から強く影響を受ける

システム設計書間の関係（トレーサビリティ）



＜システム要求分析 ＞

要求仕様はユースケースから始めて、
機能分割は方式設計で行う。

そもそも、設計の結果分かる「機能」を分類構造として、
要求仕様書を作成していくことに矛盾を感じませんか？

なぜ、要求分析をユースケースから始めるのか

MITSUBISHI SPACE SOFTWARE

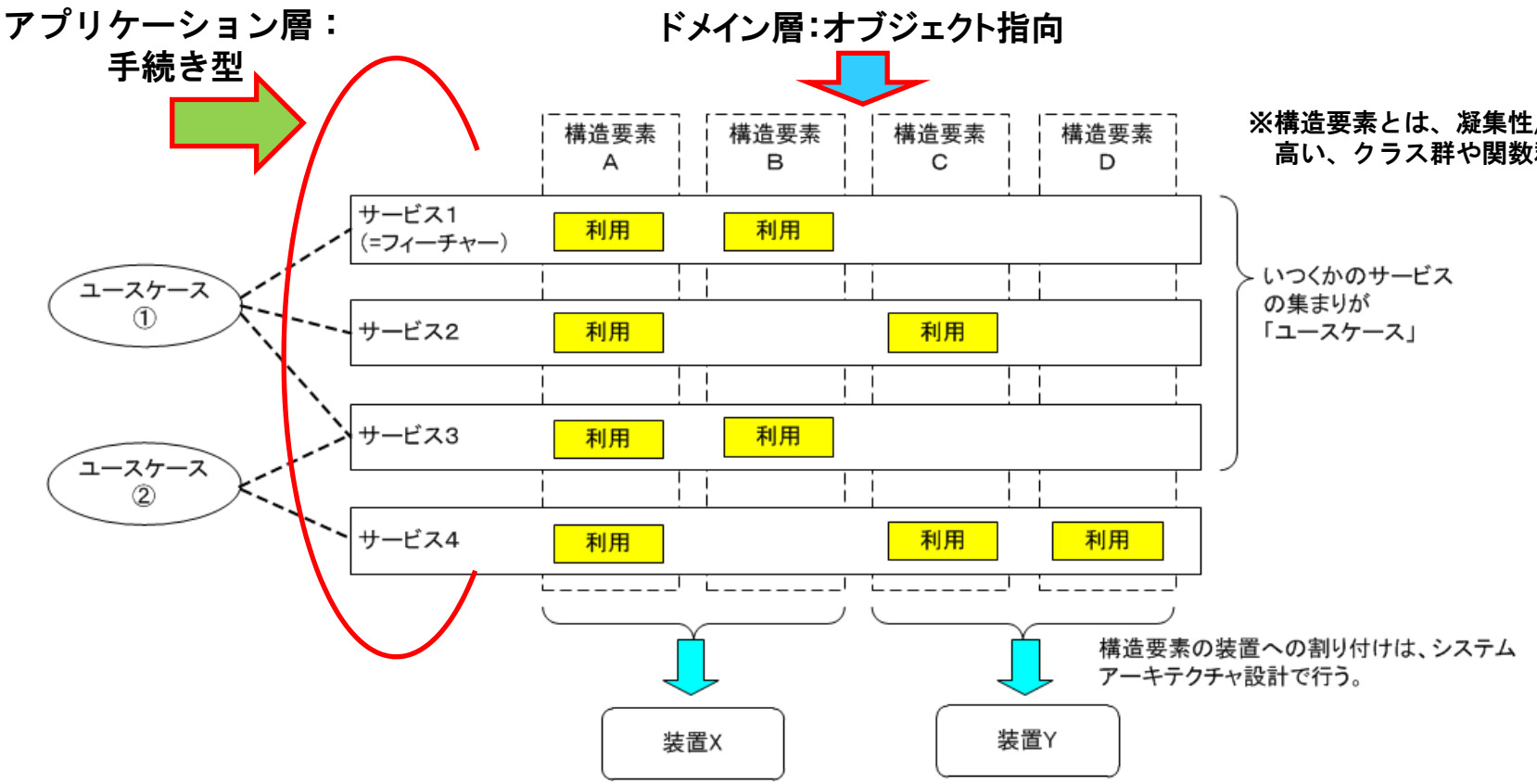
ユーザ要求とは、「システムが提供するサービス」のことであり、それは、複数のシステム機能の組合せで構築されるものである。したがって、方式設計の結果として分かる機能を要求定義の粒度にすること自体が矛盾する。

■サービスと機能は、それぞれ、次のように定義される。

・機能　　＝ 入力から出力への関数

・サービス　＝　Σ(機能)　：　アクターにとって価値ある粒度になるまで機能を集めたもの

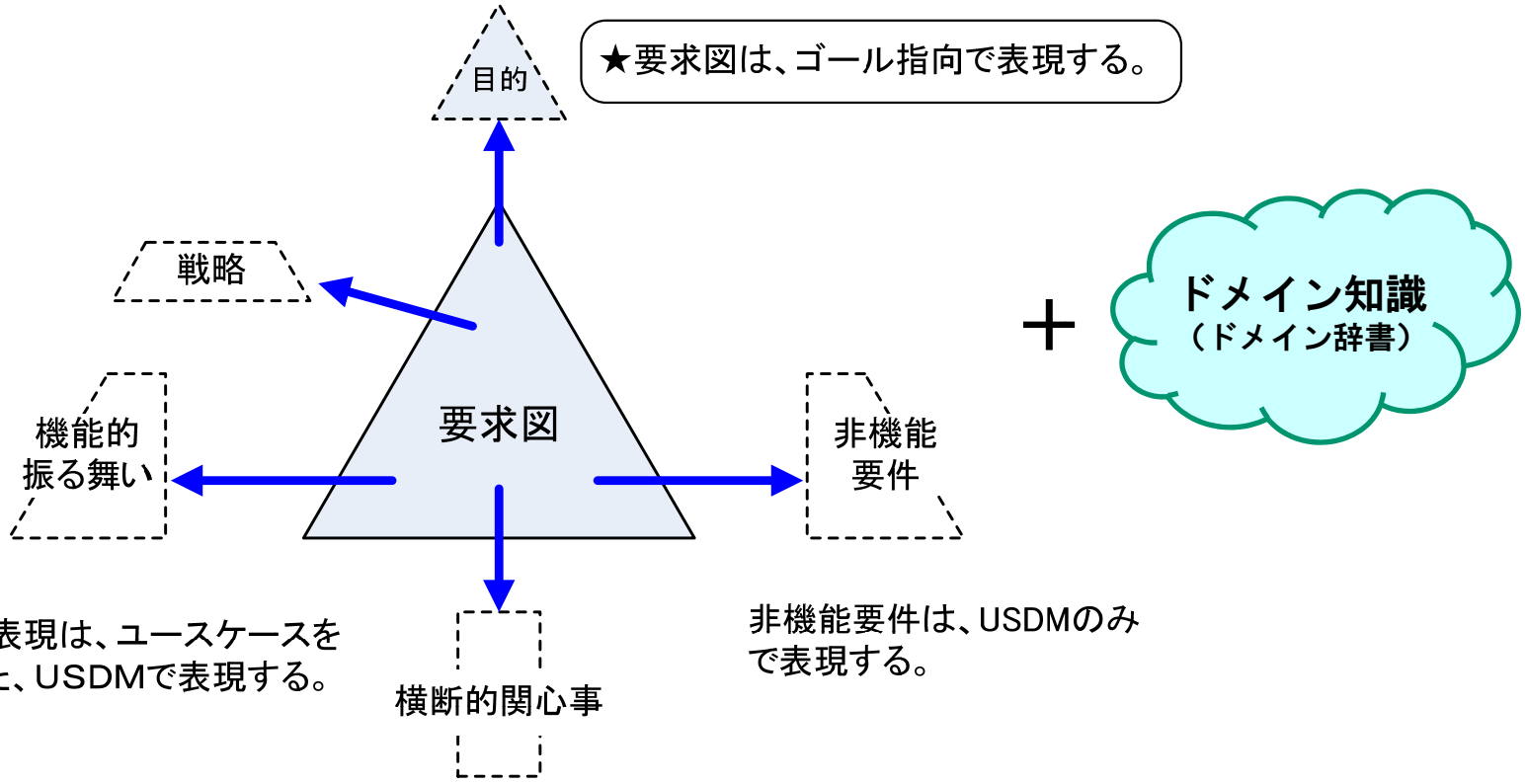
ユースケースとサービス(アプリケーション層)、機能(ドメイン層)とは、下図のような関係で表現される。



組込システムの要求分析

MITSUBISHI SPACE SOFTWARE

要求図では、システムに関わる利害関係者が求める、システムが持つべき能力や満たすべき条件をツリー構造で記述する。それは、①目的、②戦略、③機能的振る舞い、④横断的関心事、⑤非機能要件から構成する。ただし、③機能的振る舞いは、ユースケースで表現するので、実質的内容は、そちらに任せ、⑤非機能要件も、USDMの表形式で表現するので、実質的に残る要求内容は、横断的関心事のみとなる。そして、この要求図にドメイン知識を加えた物が、システム要求仕様である。



要求を仕様化する方法が必要

ユースケース記述だけでは仕様を拾いきれない！！

要求定義 （視点： システム外部）	◆「～がしたい」 利用者の希望(Require)。 ◆事業運用をビジネスレベルで考え、それを実現するコンピュータシステムへの要求。機能要求は「システムの外側からみた振る舞い」で記述できる。 ＜ユースケース記述は、要求を運用面から(ダブりがあっても)漏れなく表現する方法である＞	◆本を検索したい
仕様定義 （視点： システム内部）	◆「～が必要」要求の目的を満足する機能や条件 ◆システムが要求を満たすべき“具体的な振る舞い”を記述したもの。 ◆機能の程度やDB・通信などの利用方法。仕様は要求に含まれる“動詞”と “目的語”である。 ＜ USDM と考えて良い ＞	◆素早く探す（1秒以内） ◆あるテーマに関する本をできるだけ多く探す（さらに仕様は詳細化要）

ユースケース記述とUSDMの関係

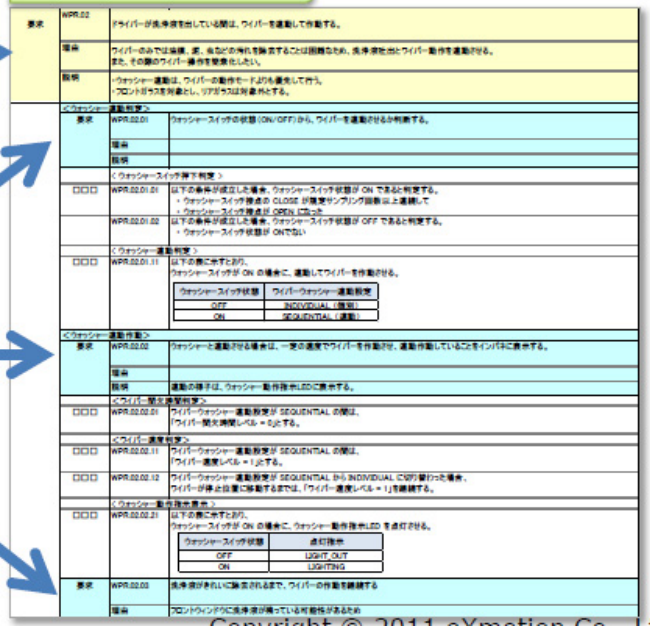
- ユースケース全体をUSDMの**上位要求**に展開
- ユースケースの各ステップはUSDMの**下位要求**あるいはグループに展開
- ユースケースの各ステップの実現方式をUSDMの**仕様**に展開

システムの外側 ⇐ ⇒ システムの内側

ユースケース記述

ユースケース番号		UC-WPR-02
ユースケース名		ウォッシャーと連動して、ワイパーを作動させる
概要		ドライバがウォッシャーを使ってフロントウインドウを洗浄している場合、 洗浄液が吐出している間は、システムはワイパーを作動させる。 油膜、泥、虫を洗浄除去する
主アクタ		ドライバ
副アクタ		・ワイパーアクチュエータ ・洗浄液吐出ポンプモータ ・ウォッシャーSW（洗浄液吐出スイッチ、PushSW2） ・洗浄液吐出ポンプモータの動作を模擬するためのLED(LED1)
事前条件		ウォッシャーSW を操作
事後条件		ワイパーの作動が停止している
フロー	STEP	ステップ
基本フロー(B)	1	ドライバは、ウォッシャーSW を操作して、フロントウインドウに洗浄液を吐出する。
	2	システムは、洗浄液が吐出している間、ワイパーを作動させる。 <A1><E1><♦2>
	3	ドライバは、ウォッシャーSWの操作を止め、洗浄液の吐出を止める。
	4	システムは、洗浄液がきれいに除去されるまでワイパーの作動を継続する。
	5	システムは、ワイパーの作動を停止させる。 停止位置まで動作させ停止、または停止位置まで動作させUC01に復帰 （ユースケースの終了） <A2>

USDM



ユースケースは、”システムの外側”から見える振る舞いをアクターの視点で表したものである。

アクターは、〇〇を行う →

＜ 機能要求 ＞

システムは、▲▲を行う ←

機能仕様 ＝ システム内部の振る舞い

【注】(株)エクスモーションで実践している手法

システム方式設計

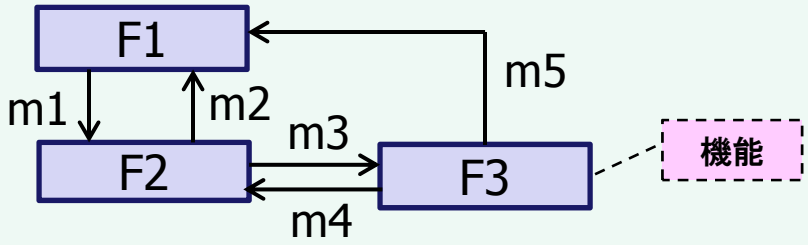
システム方式設計で行うこと

システム方式設計では、「ハードウェア／ソフトウェアの役割分割を主目的」に下記事項を明確にする。

- ◆ システムの振る舞い(ユースケース)から“大きな”機能への変換(論理アーキテクチャ)。
- ◆ 論理アーテキテクチャから物理アーキテクチャへの変換(CPU毎の機能アサイン)。
- ◆ 明確な実行プロセス(タスク)が定義できる場合、その役割定義と機能割当。

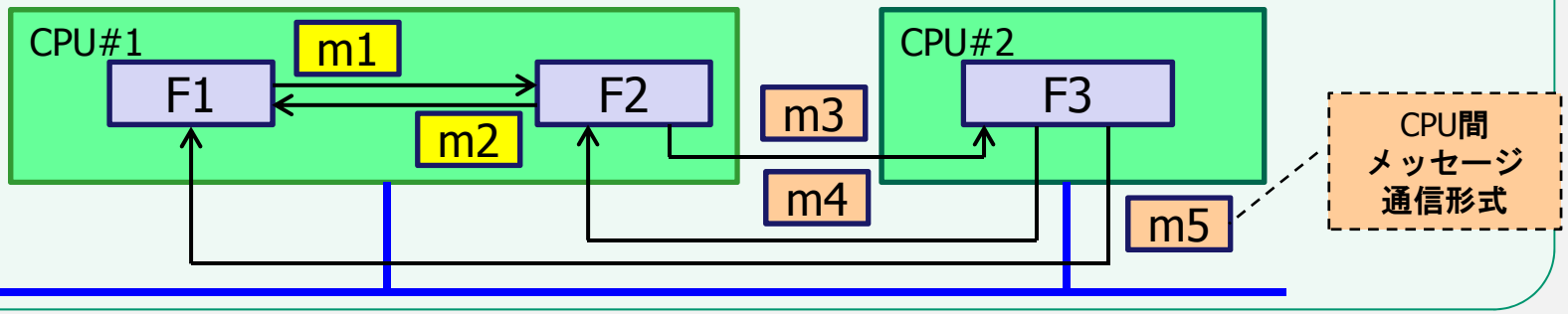
●論理アーキテクチャ

プラットフォームや実装に依存しない機能分割のアーキテクチャ



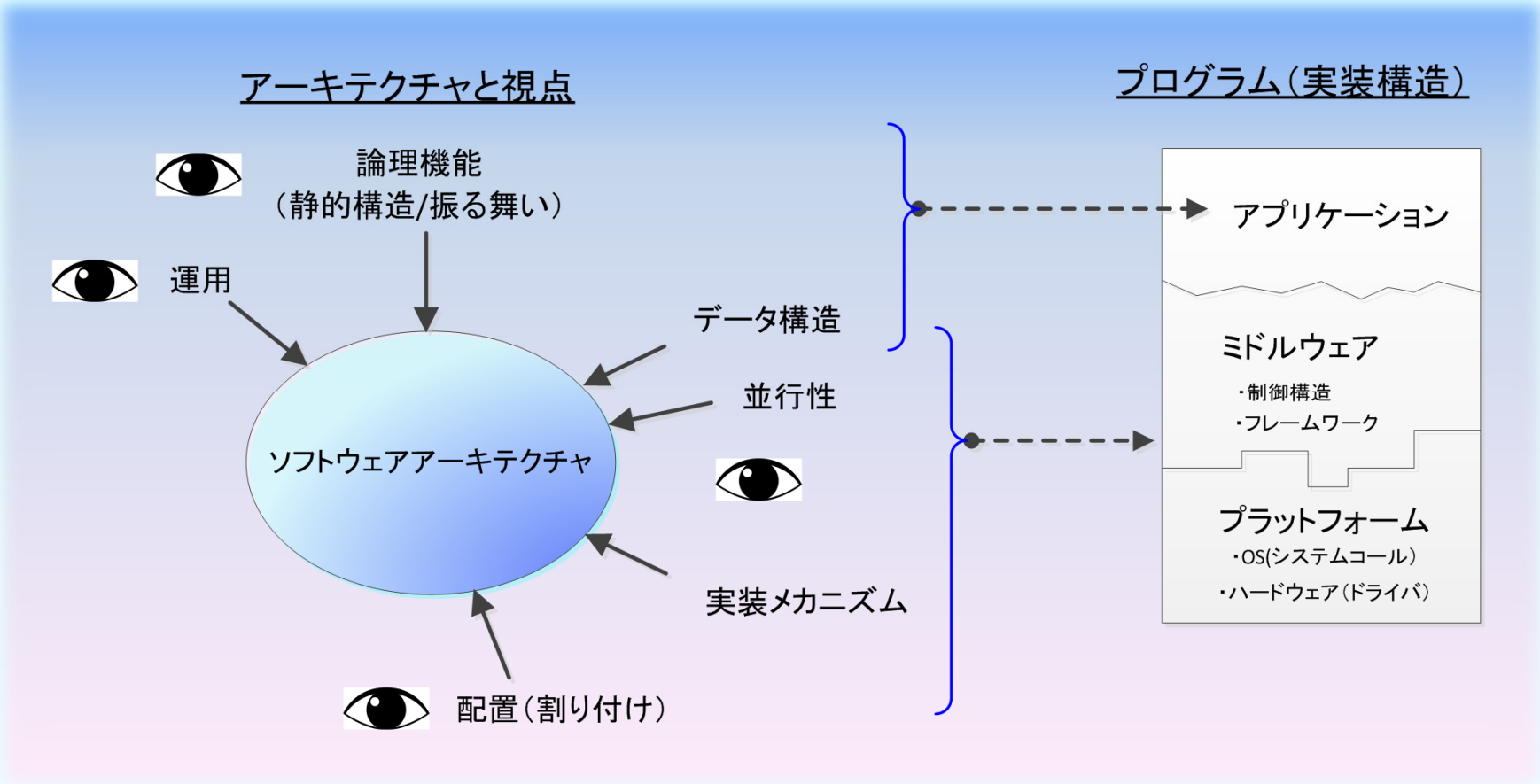
●物理アーキテクチャ

・非機能要件(性能、信頼性、保守コスト)を満たす論理アーキテクチャからの変更構造
・想定プラットフォームへの機能アサイン



アーキテクチャとは

複数の視点（ビュー）からソフトウェアを見た時の構造の合成がアーキテクチャである。



＜ドメイン分割＞

ドメイン辞書を使ってドメインを分割する。

まず、ドメイン分割

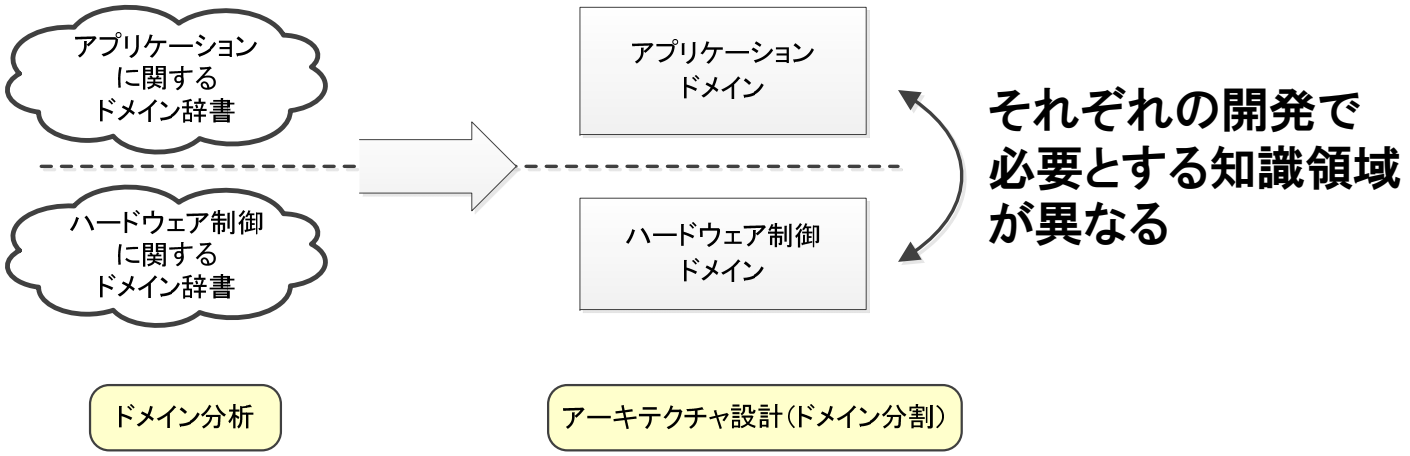
システムをドメイン単位に分割する。ドメイン分割は意味を同じくする概念の変化点の分割であり、機能分割ではない。下記に、電動ブレーキの機能分割とドメイン分割の違いを示す。

		機能分割 ⬇		
		手動Apply	手動Release	自動Release
ドメイン分割 ➡	アプリケーション			
	ハードウェア制御			

それぞれ取り扱う専門用語、知識が異なるドメインに分割することで開発に必要な知識を絞ることができ、効率よく開発が進めることができる。

例えば、アプリケーションドメイン開発者は、外部環境やハードウェア制約といった複雑なロジックを意識することなく開発ができる。一方、ハードウェア制御ドメインは、ハードウェアの振る舞いを知り、その制御方法に関する知識を豊富に持つ開発者が担当しなければならない。

ドメイン分割により、ドメイン辞書のドメイン境界は、方式設計において、レイヤ分割の一部となる。



＜ 方式設計 ＞

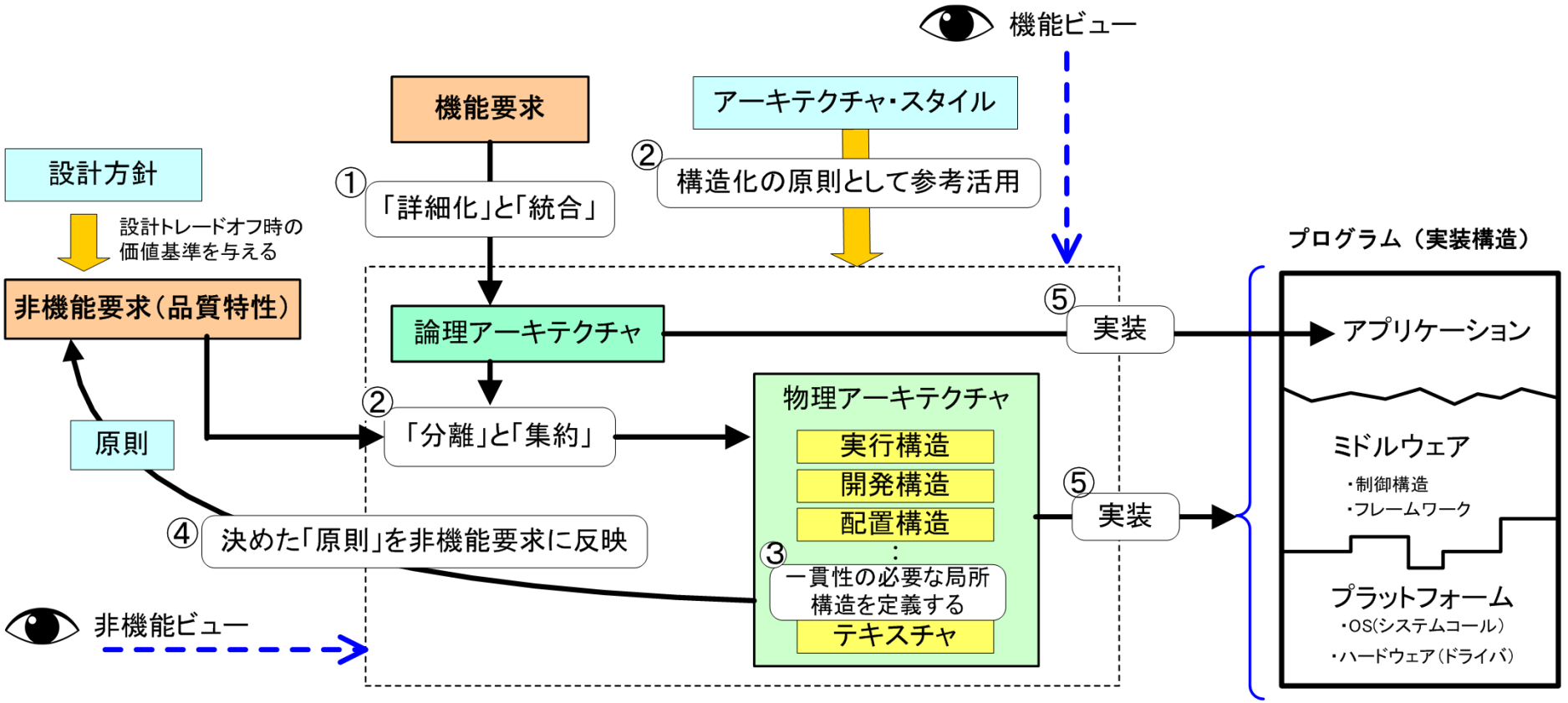
「機能要求」と「非機能要求」に対応した
設計書に分けることで、
アーキテクチャ設計の出来高を定量的に把握する。

方式設計の流れ

MITSUBISHI SPACE SOFTWARE

機能ビューと非機能ビューは、準独立関係にあり、別々に分けて設計を行うのが良い。

① 機能要求から論理アーキテクチャ(具体的な基盤に依存しない概念的な論理構造)を作成する。
② 論理アーキテクチャを踏まえて、物理アーキテクチャを設計する。この時、アーキテクチャ・スタイルを活用する。
また、構造検討において複数の選択肢が生じた際には、設計方針に照らしてトレードオフする。
③ 一貫性の必要な局所構造をテキスチャ(※)として定義する。
④ 方式設計以降の設計において、従うべき原則を非機能要求に反映して残す。



※テキスチャとは、エラー処理方法や、ユーザとのインタラクションの方法など、ソフトウェアの様々な部分に表れ、一貫性のある方針で作られる小構造である。

機能方式設計と基盤方式設計を分ける

MITSUBISHI SPACE SOFTWARE

ソフトウェア方式設計書は、それが対応する要求（機能、非機能）に応じて、大きく、論理と物理アーキテクチャの２つに分ける。

要求	方式設計		説明
機能要求	論理アーキテクチャ	機能方式設計	「SW機能要件」を満たす論理構造とそれに対応する振る舞いを設計すること。
非機能要求	物理アーキテクチャ	基盤方式設計（実装技術）	「SW非機能要件」（性能、品質等）を満たすように、論理S/W構造に対して、実装方式を設計すること。

■メリット

２冊別々に作成することで、基盤方式設計（アーキテクチャ）の出来具合を、物理量（ページ数）として把握することができる。

※保守（派生開発）の段階で、「アーキテクチャが存在しない」という声を良く聞くが、これは、機能の設計と非機能の設計が混じり合った方式設計書を１冊にまとめて作る為に、元々の設計段階で物理アーキテクチャの出来高を定量的に把握し難いことも原因の一つである。

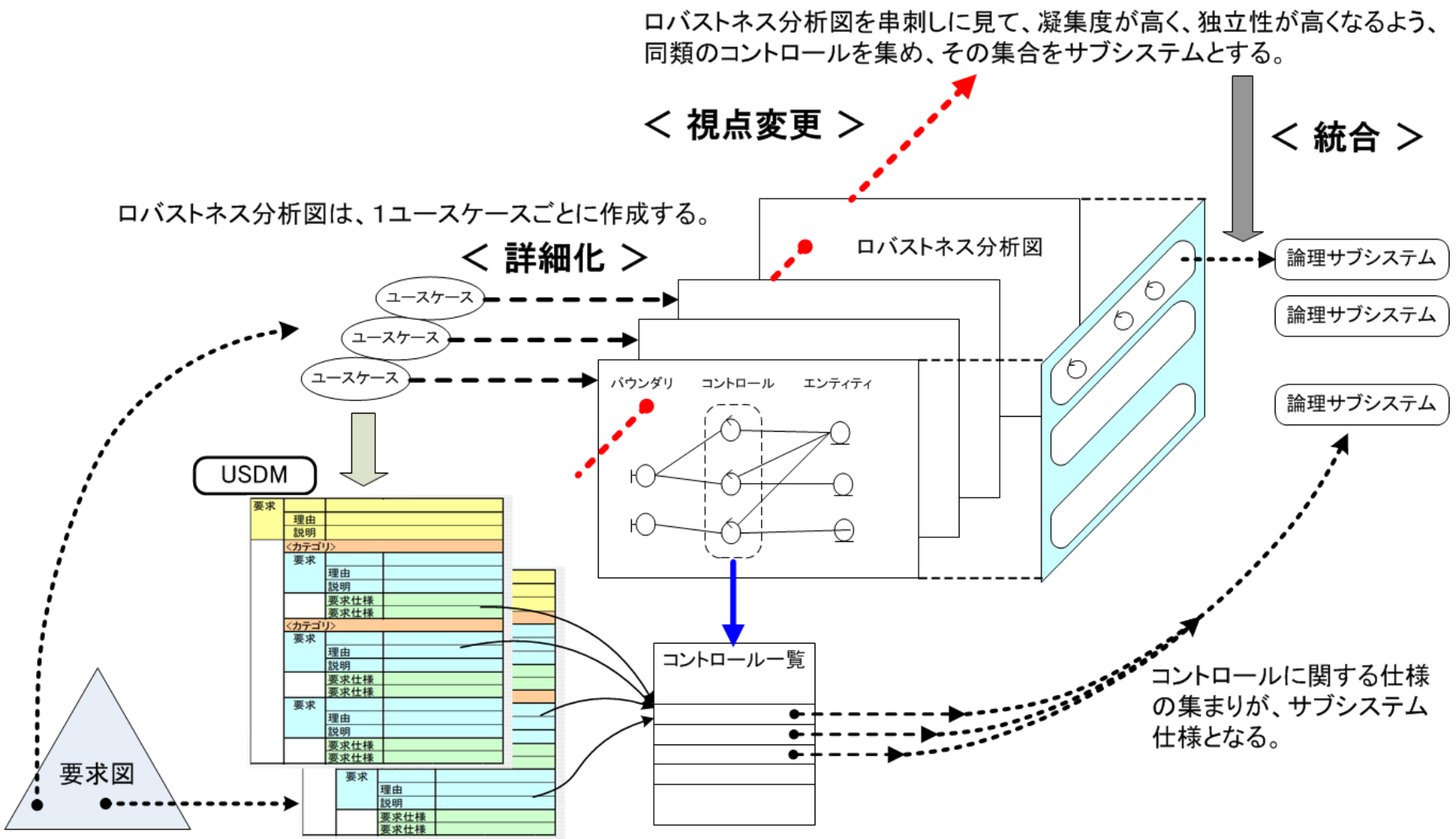
＜ 機能方式設計 ＞

機能の塊であるサブシステムを定義する。

ただし、システム方式設計は、ハードウェア/ソフトウェア境界を
分けることが最小限の作業なので、
ソフトウェア構造のサブシステム構造分割が過度にならないよう注意すること。

要求仕様分析から論理サブシステム抽出までの流れ

要求仕様(要求図他)からロバストネス分析図を作成することで、シームレスに、論理サブシステムを抽出することができる。各サブシステム抽出は、①凝集度が高く、②結合度が疎になるようにする。



DSMにより、論理サブシステムを決定する

MITSUBISHI SPACE SOFTWARE

■DSM（Dependency Structure Matrix)を使ってサブシステムを決定する

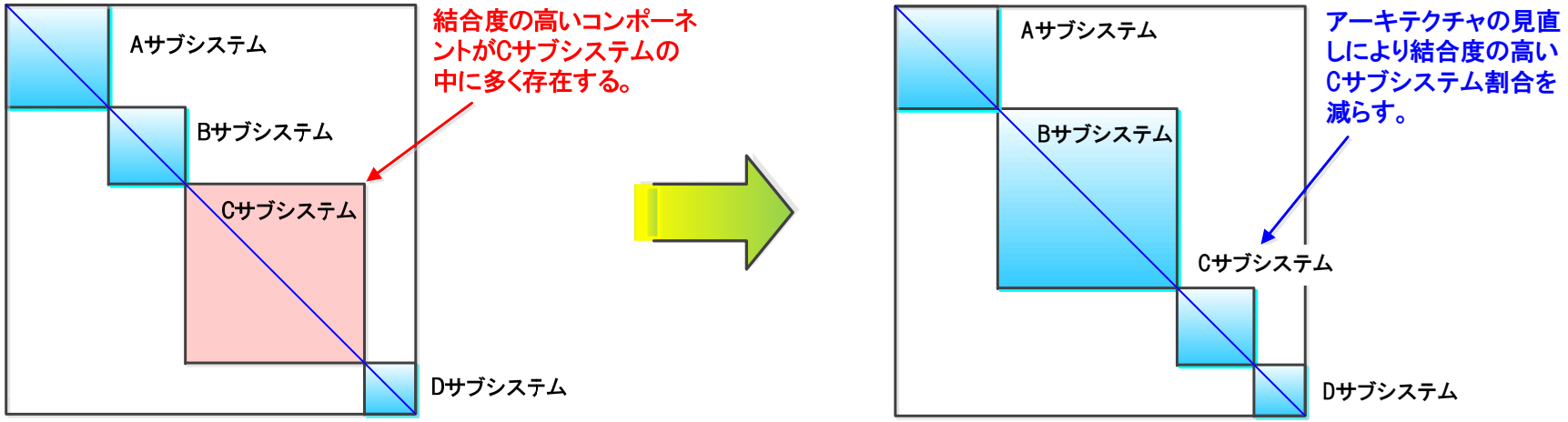
ひとつの経験的システムアーキテクチャ評価則は、「できるだけ独立したモジュール（＝外部の相互作用が少なく、内部の相互作業が多いモジュール）を選ぶ」ということである。これは、依存関係性の高い（≠複雑度が高い）構造は不具合を埋め込み易いという論文もあり、保守性を高める要求に適っている。

そこで、モジュール間の関係性を可視化するDSMを用いて、サブシステムを決定する。

	A	B	C	D	E	F
コンポーネントA		1		1		
コンポーネントB	1		3	1		
コンポーネントC		3		1		
コンポーネントD	1	1	1		2	1
コンポーネントE				2		3
コンポーネントF				1	3	

左図は、あるシステムがA,B,C,D,E,Fの５つのコンポーネントで構成されていることを示している。そして、コンポーネントAの依存関係は行/列１に表示され、コンポーネントB,Dに依存していることを表している。セルの中の数字は、コンポーネント間に設定したルールによる重みである。

■DSMによるサブシステム決定／評価例

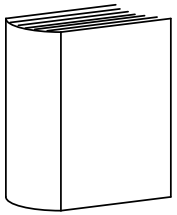


＜ 基盤方式設計 ＞

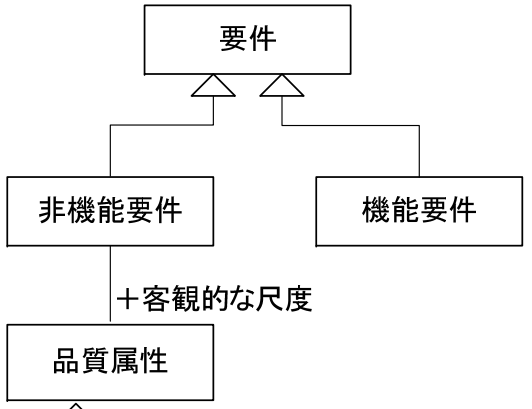
ATAM(※)で物理設計を行い、
アーキテクチャ設計のレビュー効率を上げる

（※）Architecture Trade-off Analysis Method

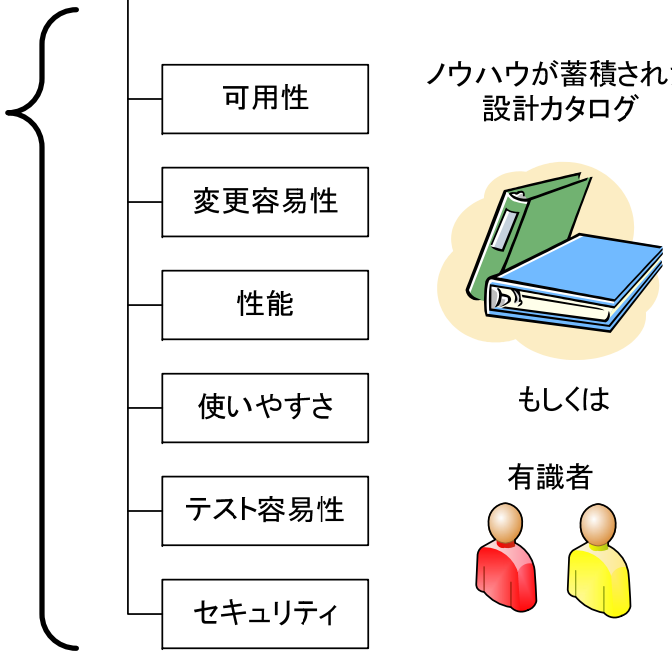
基盤方式設計とは…これがアーキテクチャである。



特に強い影響を受ける



アーキテクチャ設計に要求される特性分類	基盤アーキテクチャの記述内容	要求（品質特性）					
		可用性	変更容易性	性能	セキュリティ	試験可能性	使いやすさ
●モジュール・ビュー（構造的特性） ソフトウェアの構造	システム構成（HW,SW構成）		●				
	レイヤ構造		●				
	レイヤ間インタフェース		●				
	利用フレームワーク		●				
	論理パッケージ構成		●				
●コンポーネント・コネクタービュー（動的特性） ソフトウェアが動作する時の仕組み	プロセス管理			●			
	メモリ管理			●			
	データ管理		●	●			
	状態遷移設計および実装		●	●			
	ネットワーク管理	●	●	●			
	トランザクション管理	●		●			
	国際化		●				●
	例外処理	●					
	障害管理	●					
	セキュリティ管理				●		
	ログ管理					●	
●割り当て（配置） どの部分が誰によって作られ、どのコンピュータに配置されるか	共通ライブラリ		●				
	設計上の留意点	●	●	●	●	●	●



ノウハウが蓄積された設計カタログ

もしくは

有識者

品質特性間のトレードオフ関係

MITSUBISHI SPACE SOFTWARE

非機能要件を満足するように、論理アーキテクチャから物理アーキテクチャへの変換したい！

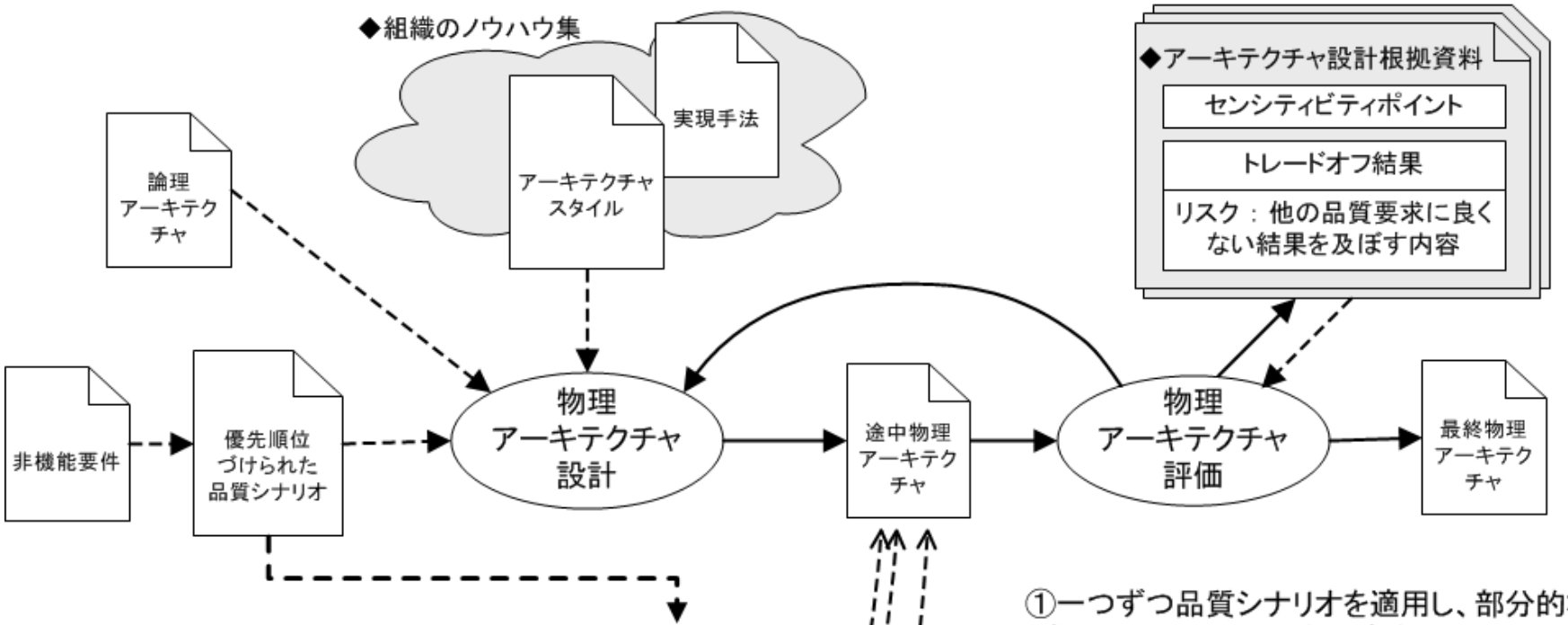
しかし、全ての品質特性をベストな状態にする万能の設計はなく、下図に示す経験的に知られている品質特性間の関係(※)をトレードオフするしかない。
例えば、ソフトウェア構造の柔軟性を高めようとすると、性能は多少犠牲にせざるを得ない。

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪
①可用性（Availability）								＋		＋	
②性能効率性（Efficency）			－		－	－	－	－		－	－
③柔軟性（Flexibility）		－		－		＋	＋	＋			
④インテグリティ（Integrity）		－			－				－		－
⑤相互運用性（Interoperablity）		－	＋	－			＋				
⑥保守性（Maintainability）	＋	－	＋					＋			
⑦移植性（Portability）		－	＋		＋				＋		－
⑧信頼性（Reliability）	＋	－	＋			＋				＋	＋
⑨再利用性（Reusability）		－	＋	－	＋	＋	＋	－			
⑩堅牢性（Robustness）	＋	－						＋			＋
⑪使用性（Usability）		－								＋	

（判例）＋　：　対応する行の特性を上げると列の特性にプラスの影響を与える。
　　－　：　対応する行の特性を上げると列の特性にマイナスの影響を与える。
　　空白：　ほとんど影響を与えない。

（※）IPA SEC編：要求工学・設計開発技術研究部会　非機能要求とアーキテクチャWG2006年度報告書

論理アーキテクチャから物理アーキテクチャへの変換



非機能要件の実装検討項目			非機能要件（品質属性）					
ビューポイント			可用性	変更容易性	性能	セキュリティ	試験可能性	使いやすさ
●コンポーネント・コネクタービュー（動的特性） ソフトウェアが動作する時の仕組み	情報	プロセス管理	●		●			
		メモリ管理			●			
		データ管理		●	●	●		
		トランザクション管理	●		●			
	並列性	タスク構造（機能-タスクマッピング）			●			
		プロセス間通信			●			
		状態遷移設計および実装		●	●			
		同期化と完全性			●		●	
		スタートアップとシャットダウン	●					
		タスク障害	●					●
●モジュール・ビュー	機能的	国際化		●				●
		例外処理	●				●	●

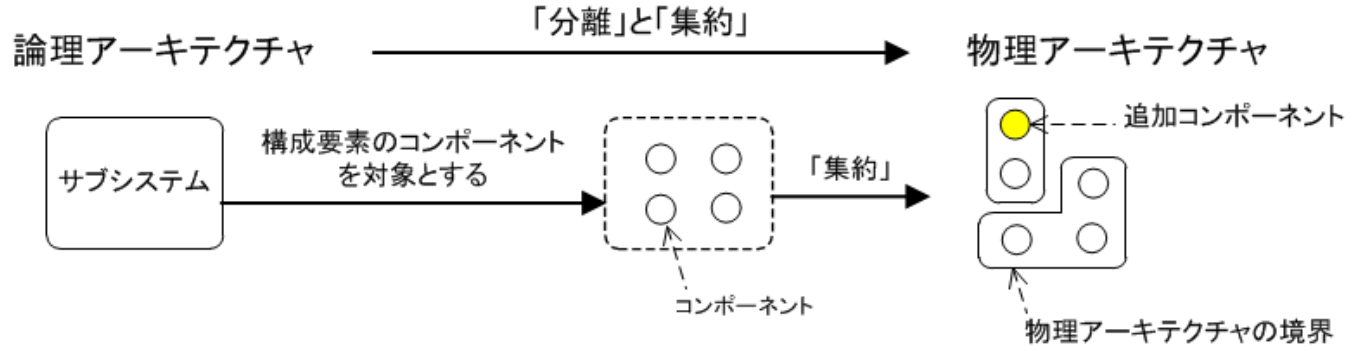
①一つずつ品質シナリオを適用し、部分的な物理アーキテクチャを作成する。
②物理アーキテクチャを一つ作る毎に、それを作った根拠資料を作成する。
③すべての品質シナリオを適用し、リスクを許容範囲内に入れられれば、終了

→ 処理の流れ

⇢ 参照

論理サブシステムから物理サブシステムへのマッピング

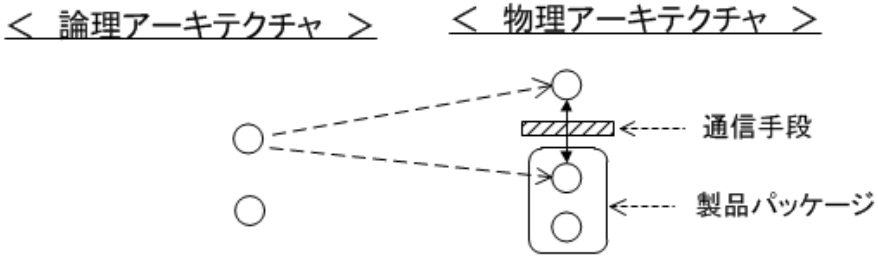
論理サブシステムから物理サブシステムへのマッピングは、通常1対1であるが、別のサブシステム枠になることもある。品質属性を考慮しながら、論理コンポーネントを、物理コンポーネントにマッピングすることで、その枠組みが決まる。



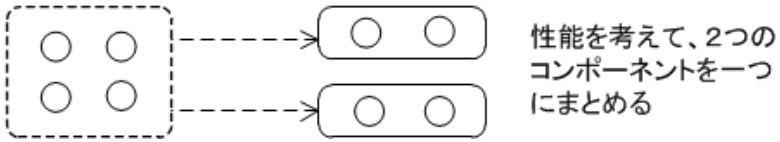
■＜論理アーキテクチャ＞　対　＜物理アーキテクチャ　＞　の関係

● ＜1＞　対　＜1＞　（　通常　）

● ＜1＞　対　＜多＞

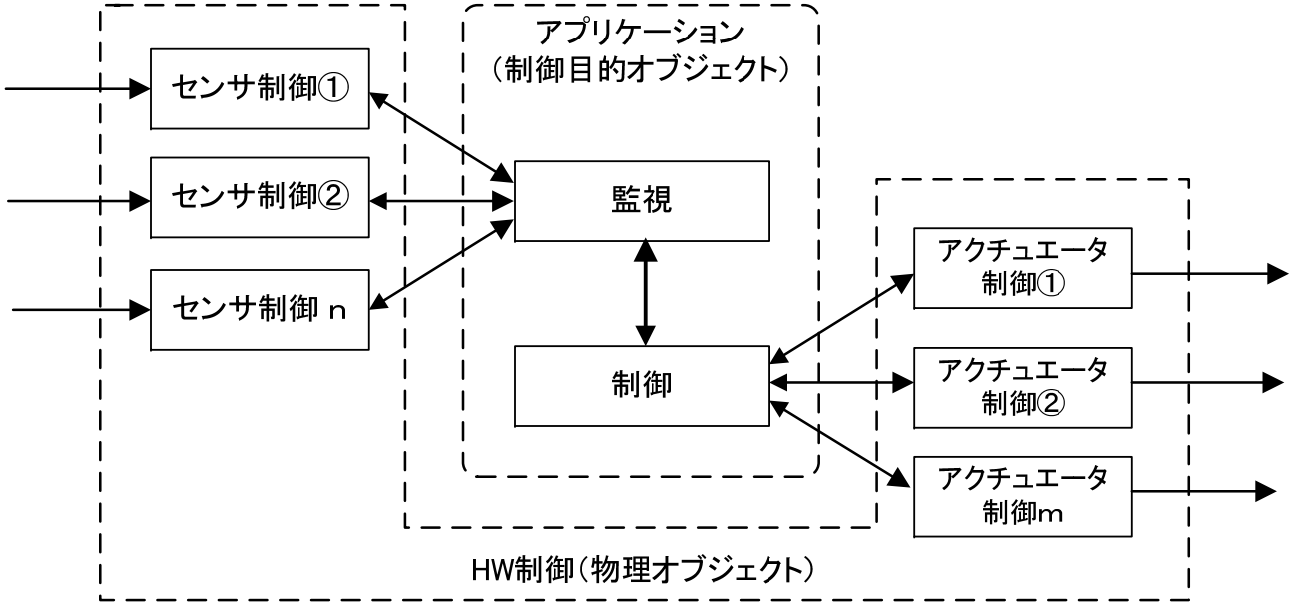


● ＜多＞　対　＜多＞



アーキテクチャ・スタイルは、複数のアーキテクチャに繰り返し現れる構造のことである。

■スタイル例：
複数センサを監視し、状況変化に応じて、アクチュエータを制御するスタイル。



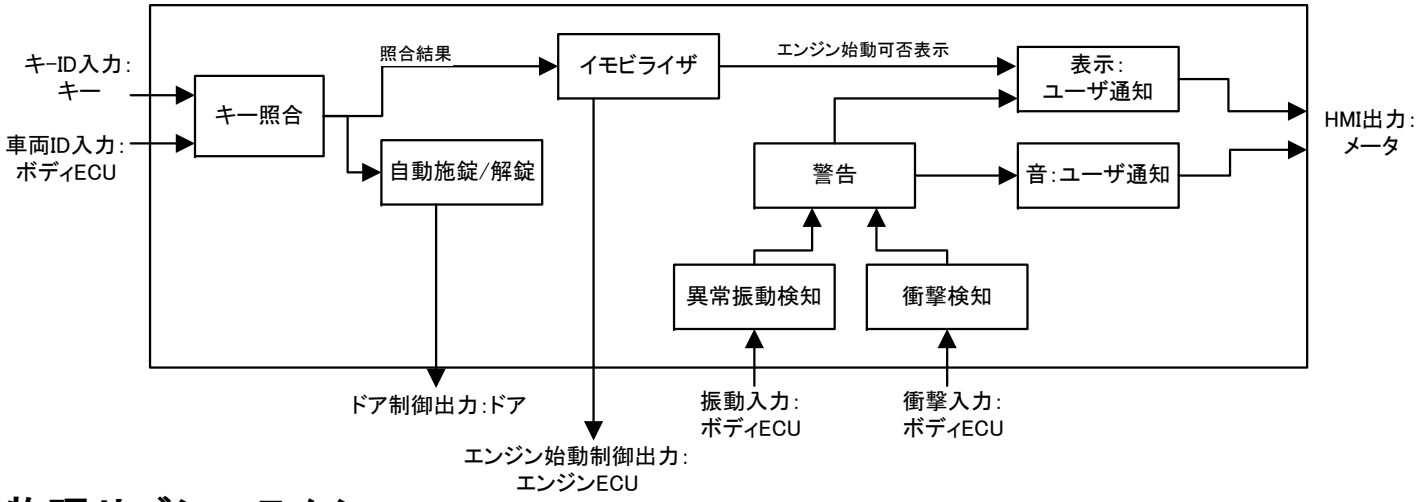
ドメイン	名称	意味	処理内容
アプリケーション	制御目的 オブジェクト	全体制御 （制御したいこと）	・制御の競合調整 ・状態遷移 ・センサ情報間の加工処理 ・センサ情報の相関関係による整合性検証
HW制御	物理 オブジェクト	制御対象機器を表現	・センサ制御（入力） ・アクチュエータ制御（出力）

論理サブシステムと物理サブシステムの変換例

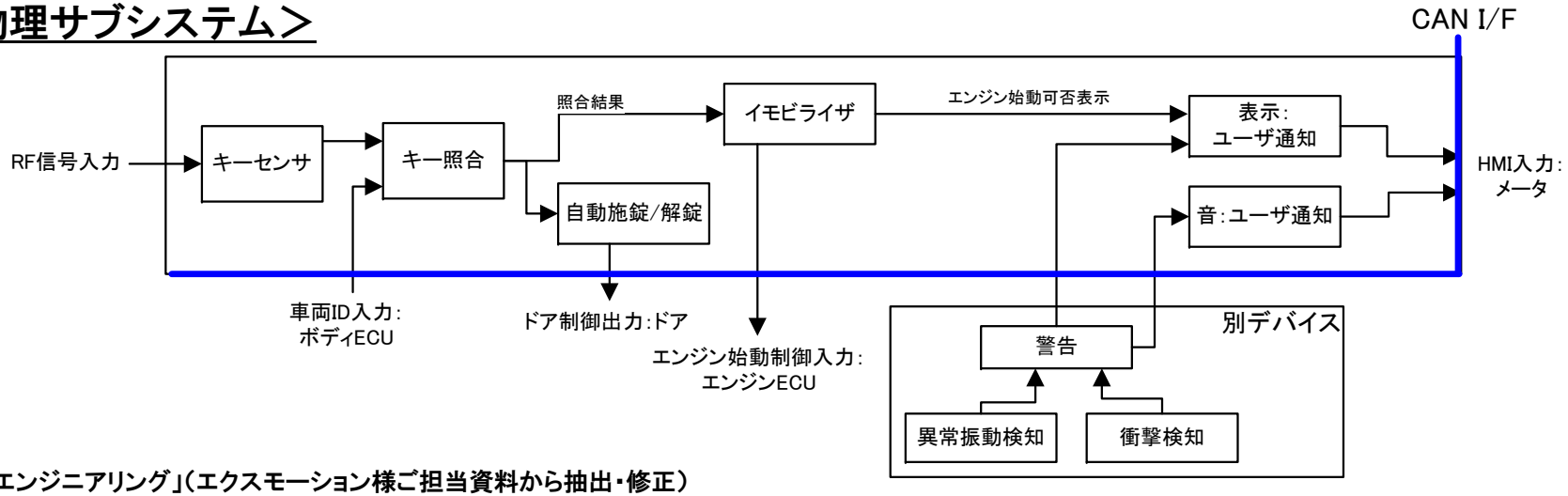
MITSUBISHI SPACE SOFTWARE

性能、開発効率、コストをトレードオフしながら、論理サブシステムから物理サブシステムに変換していく。

＜論理サブシステム＞



＜物理サブシステム＞

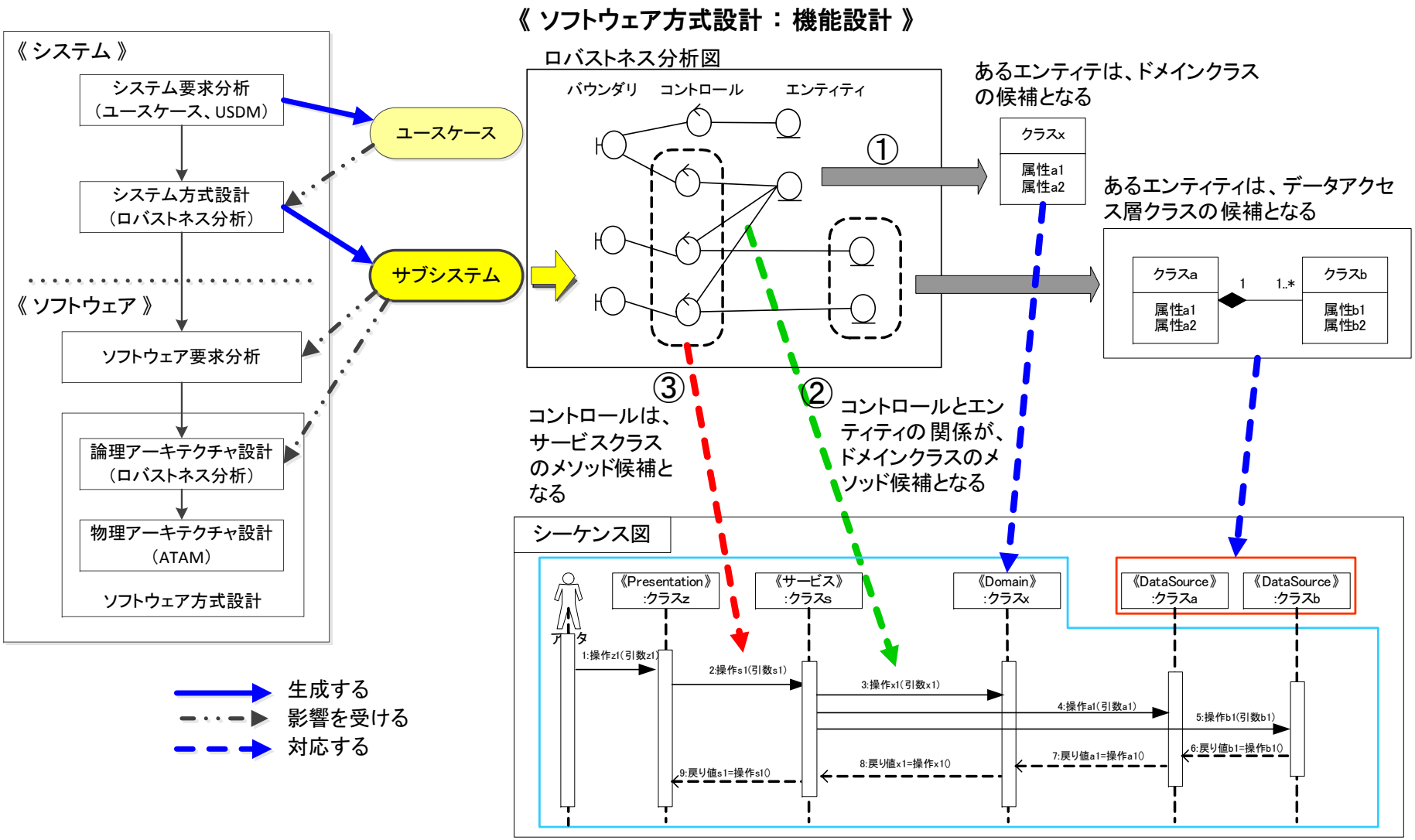


※「システムズエンジニアリング」(エクスモーション様ご担当資料から抽出・修正)

ソフトウェア要求分析、方式設計

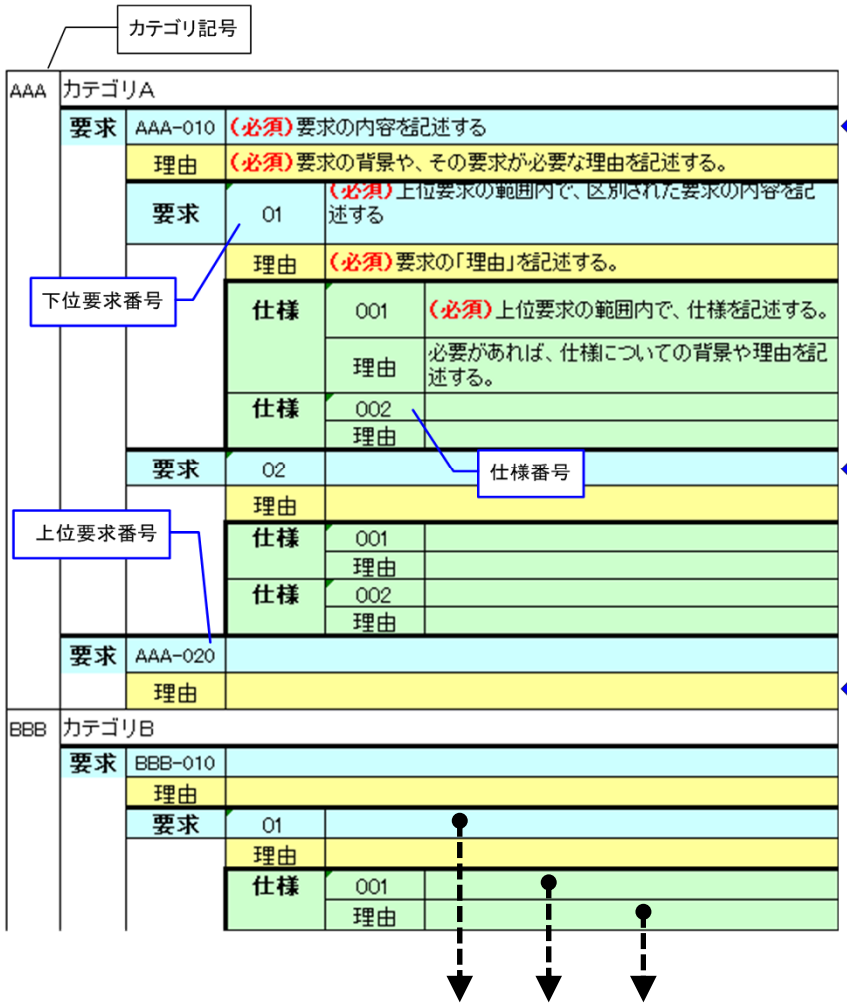
ソフトウェア方式設計

システム開発の手法は、そのまま、ソフトウェア方式設計までの手法として使える。
違いは、アウトプットの内容と粒度である。



手法の適用と定着に苦労したところ

USDMが書けない！？



■ユースケース名を最上位要求とする。

■要求と仕様を区別する。
仕様とは、要求（実現して欲しいこと）を満足すべ　き”具体的な振る舞い（手段）”の記述である。

■要求を階層化する。
要求を階層化して、個々の要求範囲を狭めて、仕様を引き出し易くする。結果、要求が漏れ難くなる。

■要求が必要な「理由」を記述する。
その理由が記述できない時は、要求ではなく、仕様が書かれていたり、要求の表現が曖昧であったりする。

（注）理由が「要求が欲しいから」と裏返し表現にならないようにすること。

■課題
文書構造は決めてくれるが…

文章（コンテンツ）が…書けない！！

→「Ｗｏｒｄ形式なら書ける」：
（本末転倒）

①開発対象システムの理解ができていない。
②文章を構造化して記述する技術がない。
③要求と仕様の区別ができない。

「文章を構造化して記述する技術がない」の対策

■対策方針

要求記述文の文法のひな形に、EARS(Easy Approach to Requirements Specification)をベースに拡張した形式で記述する。

表　EARSテンプレート構文

要求型	構文
全体説明型	<システム> は <応答>する必要がある
イベント駆動型	<望ましい条件>の<契機>が発生した場合、<システム>が<応答>する
非期待振舞型	<望ましくない条件>の<契機>が発生した場合、<システム>が<応答>する
状態駆動型	<システム>が<状態>にある限り、<応答>する
選択型	<システム>が<性質>を持つ場合、<応答>する
繰り返し	<システム>は以下の処理を<X回>繰り返す
順序指定	<システム>は以下の順序で処理を行う

■あらたな課題

SWエンジニアがEARS形式で記述すると、仕様が”プログラム処理“になりがちとなる。

⇒ 「仕様に何を書くべきか」をレビューを通じて教育する。

開発手法が組織知として構築されない！？

MITSUBISHI SPACE SOFTWARE

■課題

プロジェクト毎に、オリジナル手法を作る
（改善意欲は強いのだが、他人の良い手法を取り入れない）

■対策：トップダウンで展開

- ✓ 手法教育（座学）、理解内容を手順書として作らせる。
- ✓ 職人アーキテクトでない人に使わせる（職人は言うことをなかなか聞かない）
- ✓ 協力会社にも教育する。
- ✓ 時間を掛ける／実プロジェクト適用の失敗を手法手順、フォーマットに取り込む。

⇒ 能力向上する人（成功プロジェクト）が増える。

期間	状態
●●年～２年	着想したが、この部署の技術力では全面適用は無理と思っていたので、個人で暖めていた。
部門異動後１年	使っている開発手法とエンジニアの質を様子見。
２年目～	適用プロジェクトを一つずつ増やし、５年間の適用期間中、数々の失敗を手法にフィィードバック。部員に手法の解説書を作らせる。
部門異動	今、また、新しい部門（車載）で適用を行っている。

手法の適用の効果

業務系／組込系、さまざまな分野の実プロジェクトに適用してきた(表参照)。

表　本手法適用プロジェクト例

種別	規模	開発形態	プロジェクト説明
業務系	2M	新規	工期１年、プロセス初適用（適用そのものが要件）。
	40KL	新規	協力会社に適用
	200KL	新規	開発手法の本格適用
通信システム	600KL（1M）	流用	短工期、システム要求仕様曖昧、システム換装
車載制御機器	30KL～50KL	新規/派生	短工期、アジャイル

■効果：

- ✓ 小規模～大規模まで統一的に行える手法が確立できた(これをベースに改善継続)。
- ✓ プロジェクトの成功確率が高まった(以前なら失敗していた設計ポイントを形式化し解決)
- ✓ 初心者が理解し難い部分が分かった(教育により改善)。
- ✓ 要求仕様と方式設計の記述重複を解消できた（※1）。
- ✓ 辞書(用語検索)やサブシステム定義ツール等、手法の支援ツールが手法適用展開に有効であることが分かった(現在進行形)

（※１）派生開発学会カンファレンス2016（デンソー）ＰＦＤを活用したドキュメント再構成による開発プロセスの改善

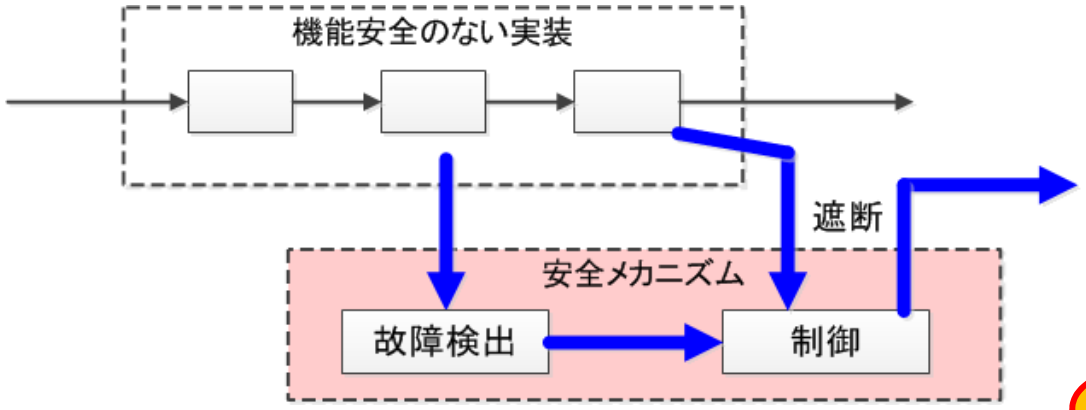
さらなる改善

機能安全への拡張

機能安全とは、

■機能安全とは 『機能により人の安全を確保する』 という考え方。
故障が発生しても安全状態に移行させる機能（安全メカニズム）を実装している。
→プロセス（A-SPICEレベル3以上）と、プロダクト（安全アーキテクチャ）を重視する

＜背景にある考え＞
- ✓ 部品単位での安全対策にいくらコストをかけても、故障を完全に無くすのは極めて困難。
- ✓ 安全メカニズムをシステムとして確実に実装することで、ユーザレベルの安全性を担保する。



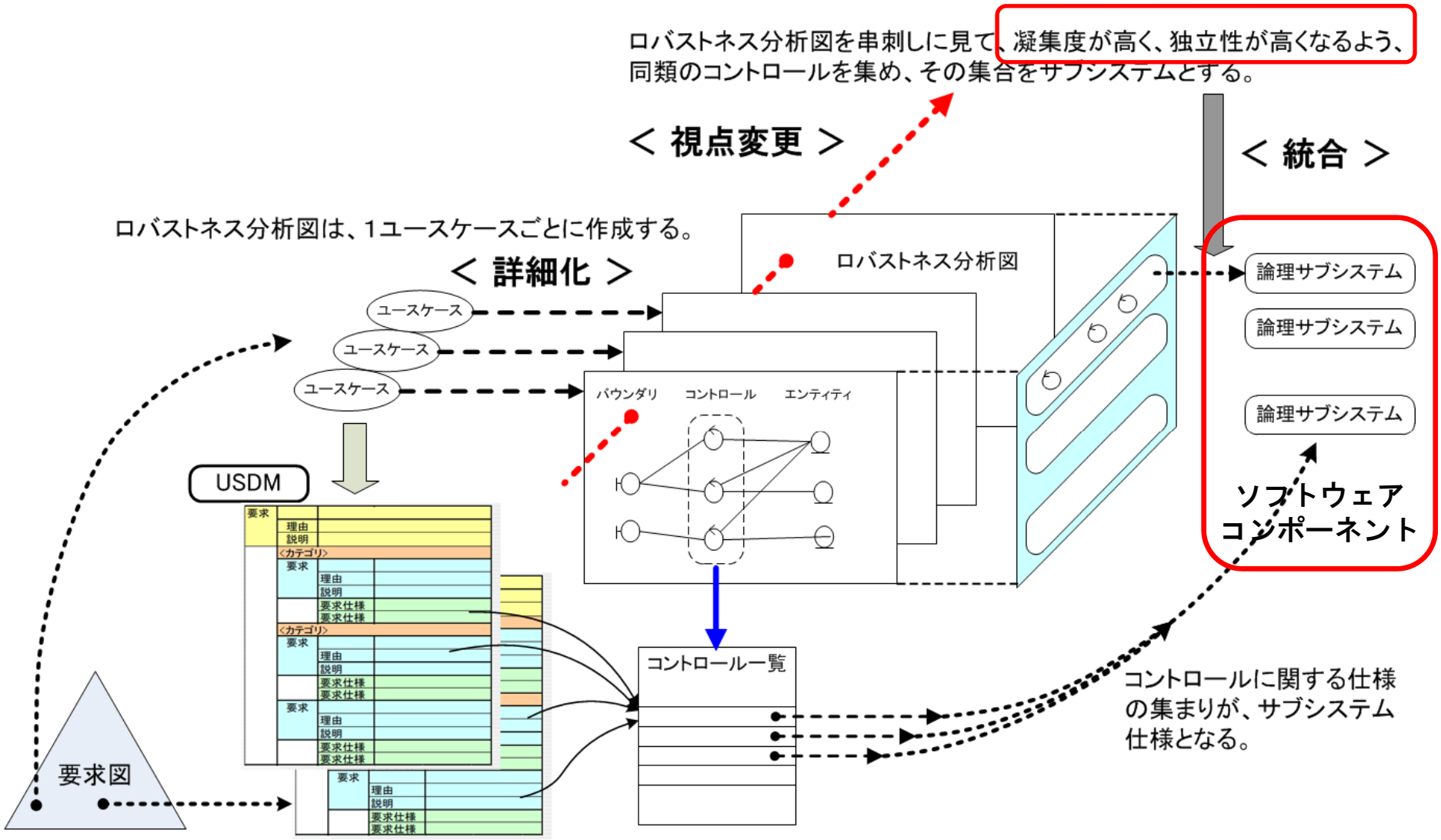
プロセス
マネージメント
だけではない！

■機能安全に必要な活動
故障しても危険状態に移行しないことを論理的に説明できること。

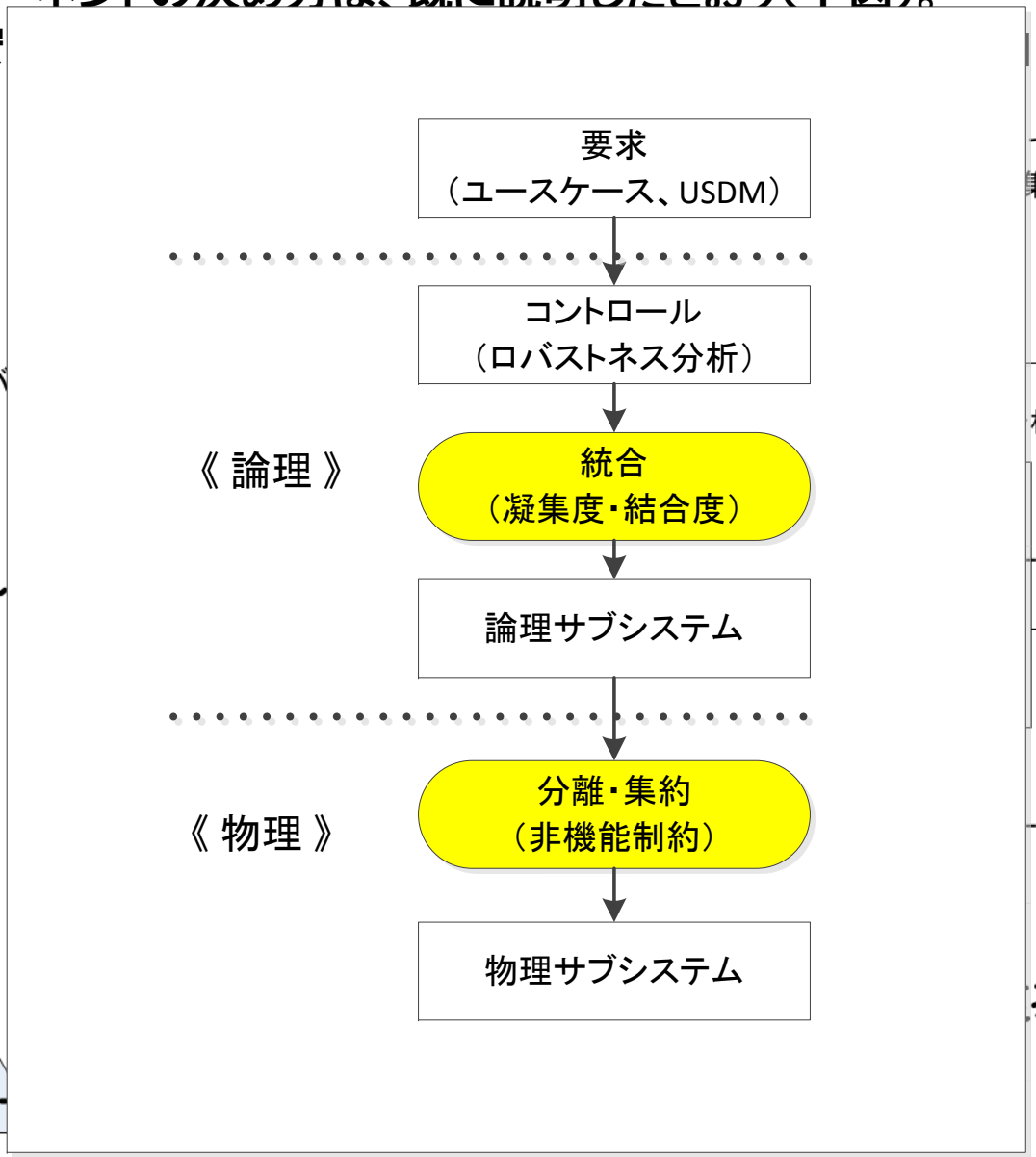
＜具体的には＞
- ✓ 科学的手法／根拠による分析、明瞭な安全系アーキテクチャ設計で安全性を実現する。
- ✓ それらを厳格なプロセス定義で確実に実施し、エビデンスやトレーサビリティで確認する。

MITSUBISHI SPACE SOFTWARE

コンポーネントの決め方は、既に説明したとおり（下図）。
機能安全規格は、「システム方式設計の科学的根拠を示せ」、と言っている。



コンポーネントの決め方は、既に説明したとおり（下図）。
機能安　　、と言っている。



て、凝集度が高く、独立性が高くなるよう、
集合をサブシステムとする。

＜ 統合 ＞

析図

論理サブシステム
論理サブシステム
論理サブシステム

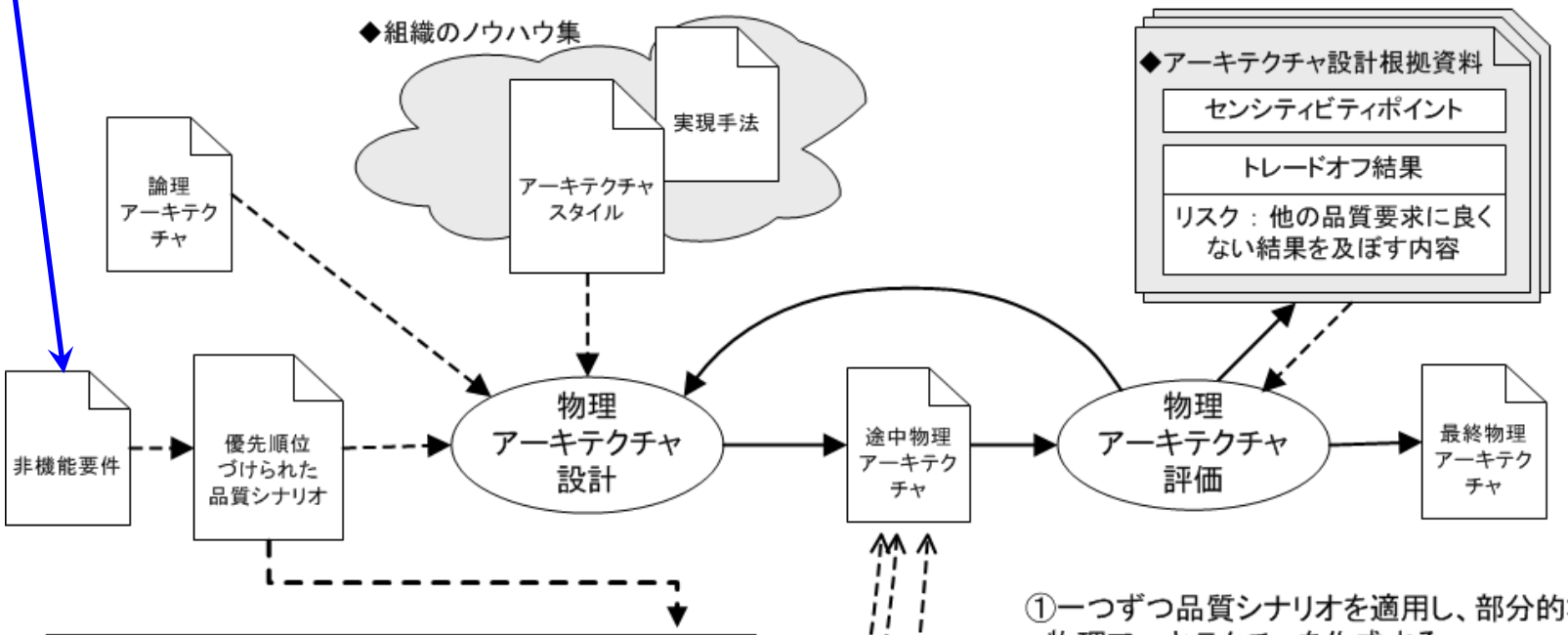
ソフトウェア
コンポーネント

コントロールに関する仕様
の集まりが、サブシステム
仕様となる。

要求図

アーキテクチャ設計（安全設計）

安全要求を最優先にすれば良い



①一つずつ品質シナリオを適用し、部分的な物理アーキテクチャを作成する。
②物理アーキテクチャを一つ作る毎に、それを作った根拠資料を作成する。
③すべての品質シナリオを適用し、リスクを許容範囲内に入れられれば、終了

処理の流れ
参照

非機能要件の実装検討項目

ビューポイント			非機能要件（品質属性）					
			可用性	変更容易性	性能	セキュリティ	試験可能性	使いやすさ
●コンポーネント・コネクタービュー（動的特性） ソフトウェアが動作する時の仕組み	情報	プロセス管理	●		●			
		メモリ管理			●			
		データ管理		●	●	●		
		トランザクション管理	●		●			
	並列性	タスク構造（機能-タスクマッピング）			●			
		プロセス間通信			●			
		状態遷移設計および実装		●	●			
		同期化と完全性			●		●	
		スタートアップとシャットダウン	●					
		タスク障害	●					●
●モジュール・ビュー	機能的	国際化		●				●
		例外処理	●				●	●

さいごに

私たちは、アーキテクチャ設計書が無い状態の派生開発であっても、
コストを考慮しながら、可能な限りアーキテクチャを作成します。

「アーキテクチャ設計書が無い」状態を生んでいることの一つに、
「アーキテクチャ設計方法を知らない」という要因もあると思います。
結局、これを解決しない限り、派生開発で、部分的なアーキテクチャ設計書
を作成しようとしても、そもそもその能力が無いということになります。

そこで、今回、スクラッチ設計で弊社が実際に行っている設計手法の一部を
ご紹介させて頂きました。

Mitsubishi Space Software

私たちには“宇宙品質”を支え続けてきた、スピリッツ・テクノロジー・ストーリーがあります。
豊富な「経験値・集合知・形式知」を活かし、お客様の課題を一緒に考えながら解決します

ご用命頂きましたら、
手法の詳細を、ご説明させて頂きます。