MITSUBISHI SPACE SOFTWARE

高信頼アーキテクチャ設計手法ATAMの実践

いざというときの安全性を担保してくれる信頼性の高いアーキテクチャを作るために。

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
  - ✓ 本手法を適用している事業の説明
  - ✓ 改善に至った経緯
- 手法の概要
  - ✓ 本手法の流れ
  - ✓ システム設計書間のトレーサビリティ
- システム方式設計
  - ✓ 本方式設計の特徴
  - ✓ 機能方式設計
  - ✓ 基盤方式設計
- 本開発手法の適用効果
- さらなる応用 – 機能安全への拡張

＜ はじめに ＞

弊社の事業分野：本手法を適用している事業説明

MITSUBISHI SPACE SOFTWARE

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）

改善に至った経緯

2004年　CMM　正式ｱｾｽﾒﾝﾄにて ﾚﾍﾞﾙ4達成を確認
2013年　Automotiv-SPICE v2.5　正式ｱｾｽﾒﾝﾄにて ENG4～8 ﾚﾍﾞﾙ4達成を確認

スクラッチ開発が少なくなり、派生開発が多くなってきた

組織品質は（表面上）良くなった。

プロセス偏重、プロダクト品質確保に必要な開発技術力維持・育成がおきざり。

スクラッチ開発力（例：アーキテクチャ設計、結合テスト）を実践の場で磨く機会が少なくなってきた。

いざ、スクラッチ開発を行うと、設計品質が極端に下がる。

外部も同じ傾向なので、社外から優秀な人材を調達することも難しくなっている（※1）。

特に、機能安全設計は、アーキテクチャ開発力が必須である。

「スクラッチ開発を形式化、手順化し、自社の組織的開発力を底上げする」
ことが最優先課題となってきた。

※1　SPI Japan2011 設計力強化全社活動（パナソニック）

＜ 手法の概要 ＞

設計の詳細化が形式化されているので、
設計を進めることで、自然とトレーサビリティが確保できる。

本設計手法の流れ（システム設計部）

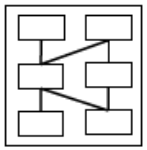
MITSUBISHI SPACE SOFTWARE

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

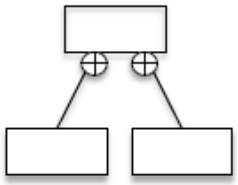
システム方式設計

①開発対象内で扱うドメイン用語の構造定義



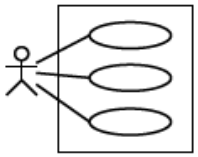
オントロジーモデルなど

②システム要求の構造化



ゴール指向要求図

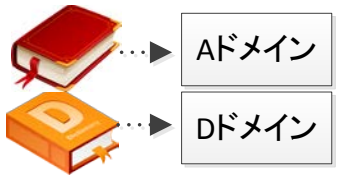
③ユースケースと具体的要求仕様



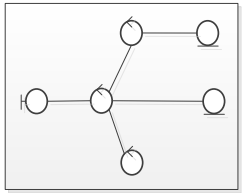
+ USDM

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

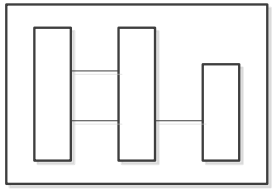


⑤ユースケースの振る舞い定義

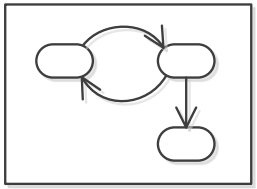


ロバストネス分析図
／シーケンス図

⑥システムを論理的に機能分割する



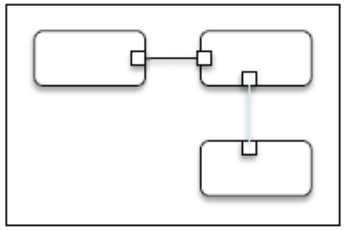
ブロック定義図



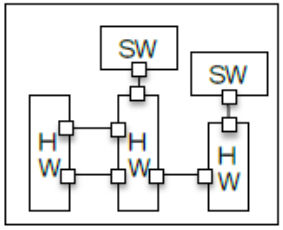
状態遷移図

等

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

等

論理アーキテクチャ

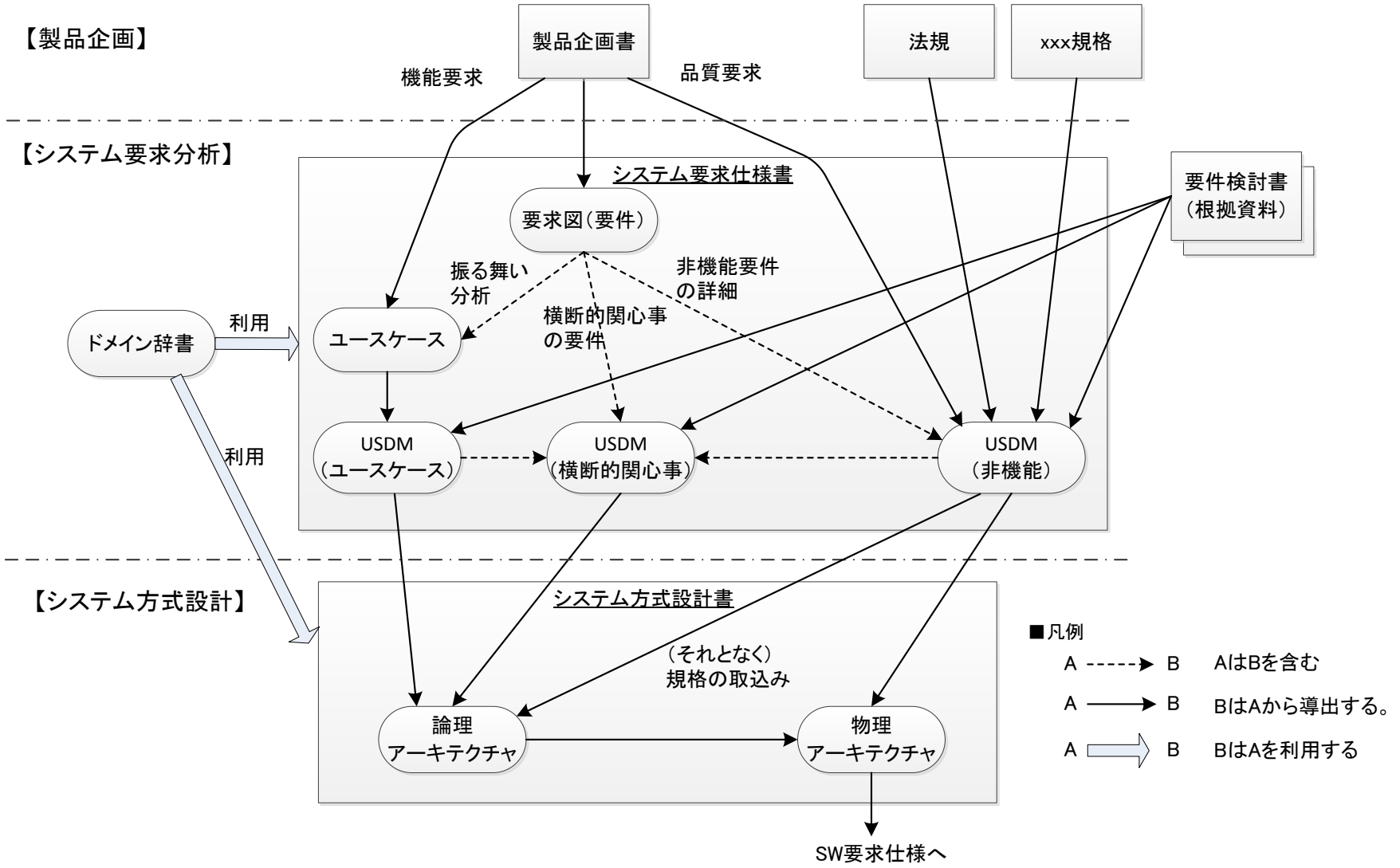
物理アーキテクチャ

【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トレーサビリティ

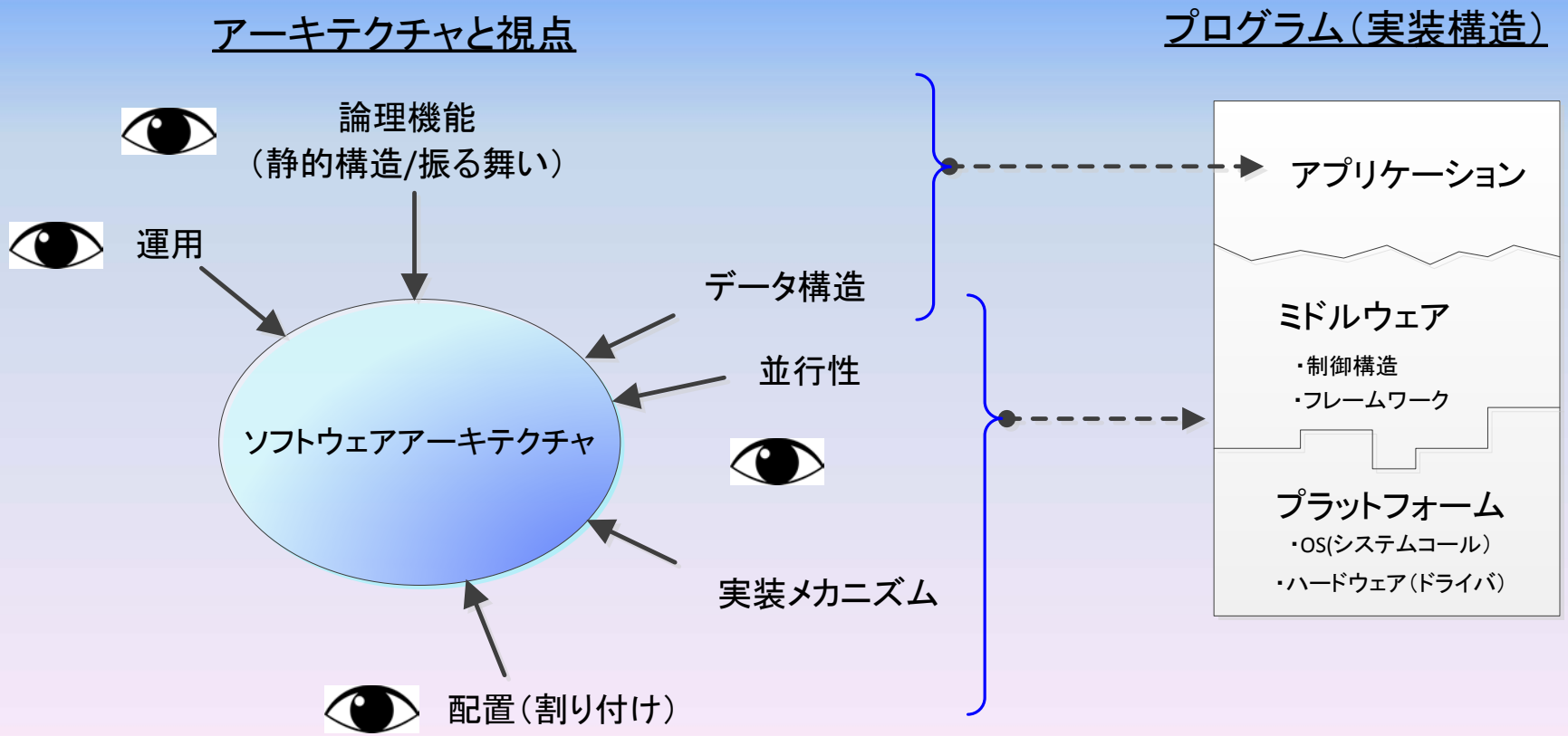
ツール利用により、要求仕様項目単位で、トレーサビリティの欠陥確認、ならびに、設計の濃淡確認が自動可能。



システム方式設計

アーキテクチャとは

複数の視点（ビュー）で、システムやソフトウェアの構造・振る舞いを表現したものがアーキテクチャである。



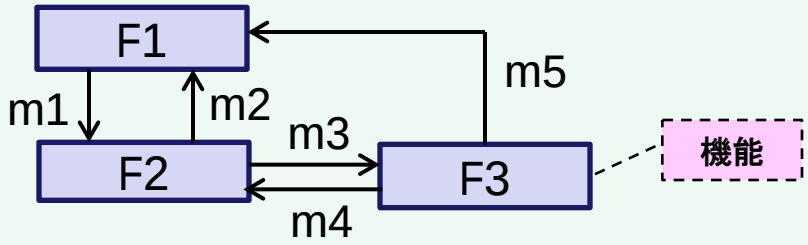
システム方式設計で行うこと

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。

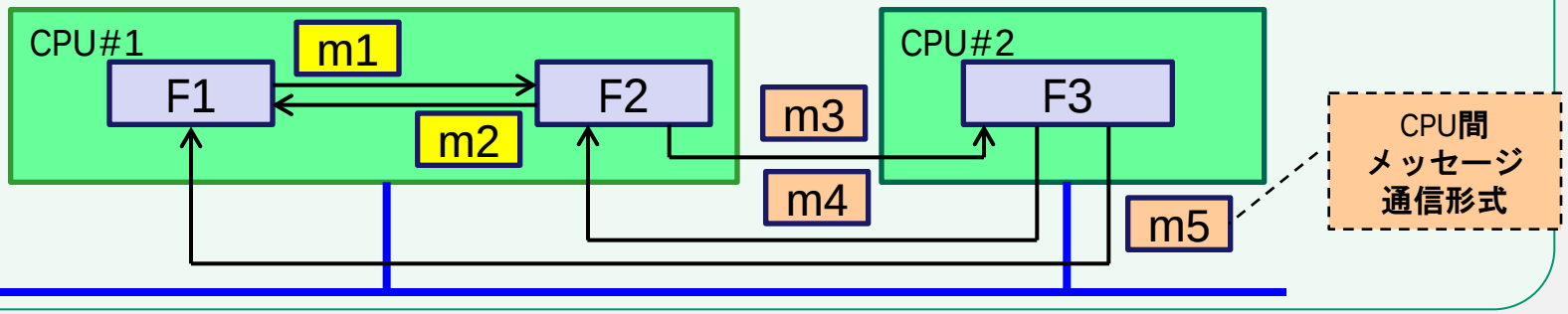
●論理アーキテクチャ

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



＜ 本方式設計の特徴 ＞

「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・質を定量的に把握できる。

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	機能方式設計	論理アーキテクチャ	「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	基盤方式設計（実装技術）	物理アーキテクチャ	「SW非機能要件」（性能、品質等）を満たすように、論理アーキテクチャに対して、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。

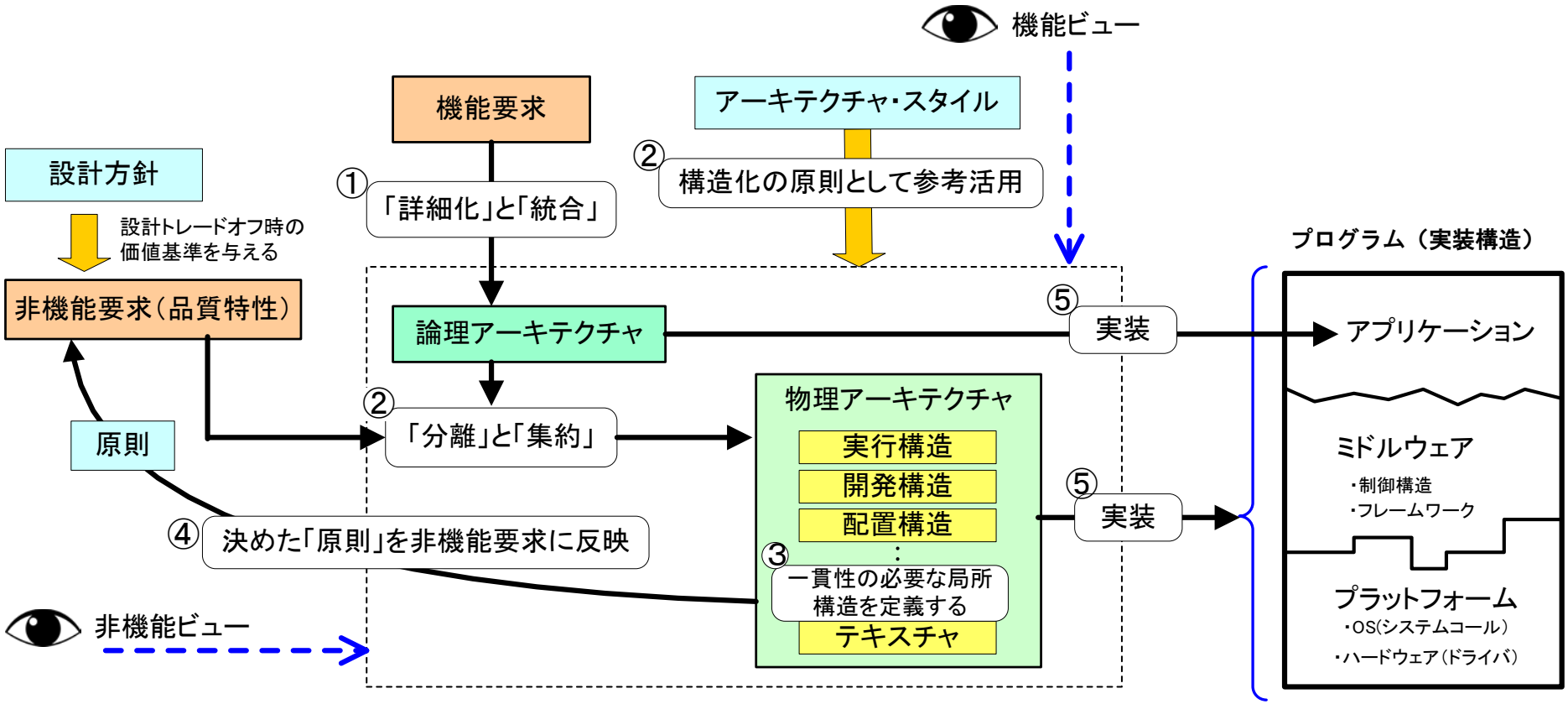
※保守（派生開発）の段階で、「アーキテクチャが存在しない」という声を良く聞くが、これは、機能と非機能の設計が混じり合った方式設計書を１冊にまとめて作る為に、設計段階で物理アーキテクチャの出来高を定量的に把握し難くしていることも原因の一つである。

方式設計の流れ ： 論理と物理を分けて設計する。

MITSUBISHI SPACE SOFTWARE

機能要求と非機能要求は、準独立関係にあり、適用を分けて方式設計を行う（IEEE1220)。

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 ＞

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。

機能方式設計の位置づけ

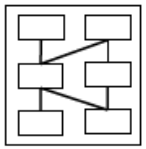
MITSUBISHI SPACE SOFTWARE

機能方式設計を赤枠（黄色）に示す。

システム要求分析

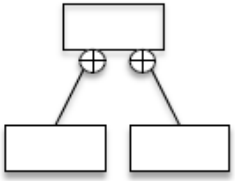
システム方式設計

①開発対象内で扱うドメイン用語の構造定義



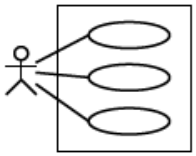
オントロジーモデルなど

②システム要求の構造化



ゴール指向要求図

③ユースケースと具体的要求仕様



+ USDM

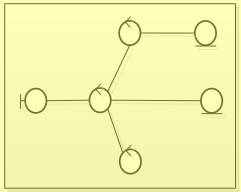
ユースケースモデル
／システムコンテキスト図

④ドメイン分割

Aドメイン

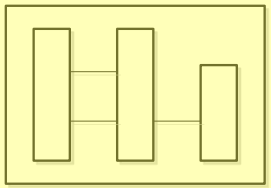
Dドメイン

⑤ユースケースの振る舞い定義

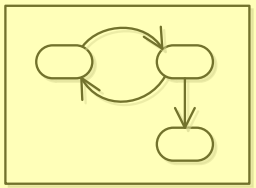


ロバストネス分析図
／シーケンス図

⑥システムを論理的に機能分割する



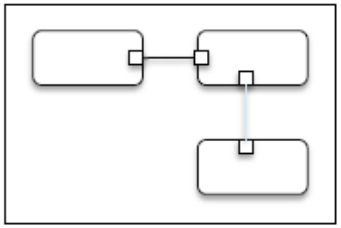
ブロック定義図



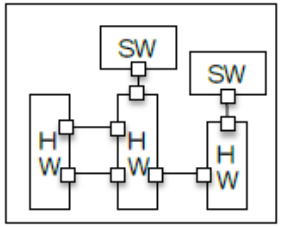
状態遷移図

等

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

等

論理アーキテクチャ

物理アーキテクチャ

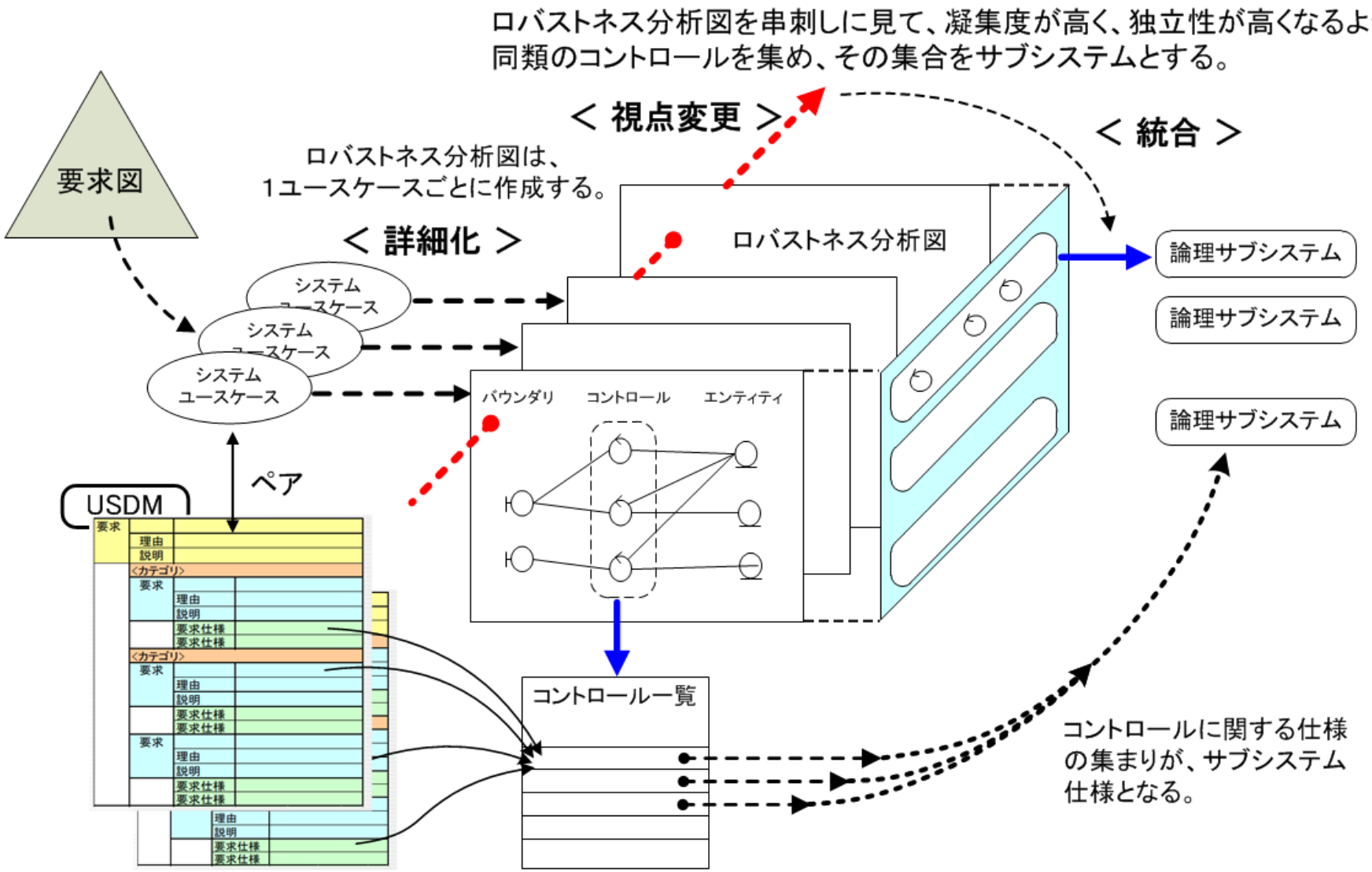
【凡例】

→ 設計分割境界に関して、矢印元から強く影響を受ける

要求仕様分析から論理サブシステム抽出までの流れ

MITSUBISHI SPACE SOFTWARE

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

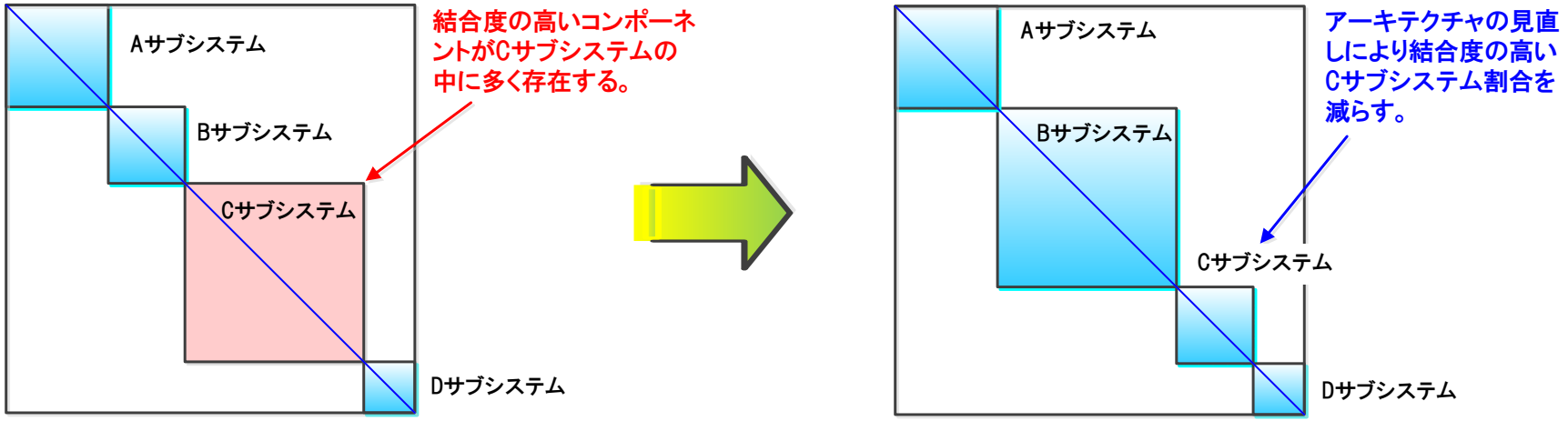
■DSM (Dependency Structure Matrix)を使ってサブシステムを決定する

「独立したモジュール群（＝ 外部との相互作用が少なく、内部間の結合度が高いモジュール群）を選ぶ」という経験的システムアーキテクチャ評価原則を使って、サブシステムを決定する。具体的可視化手法として、DSMを用いる。

	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。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・設計品質を上げる

（※）Architecture Trade-off Analysis Method

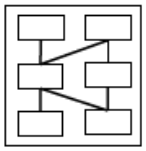
基盤方式設計の位置づけ

基盤方式設計を赤枠（黄色）に示す。

システム要求分析

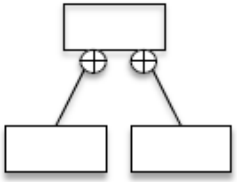
システム方式設計

①開発対象内で扱うドメイン用語の構造定義



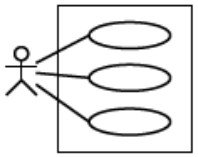
オントロジーモデルなど

②システム要求の構造化



ゴール指向要求図

③ユースケースと具体的要求仕様



+ USDM

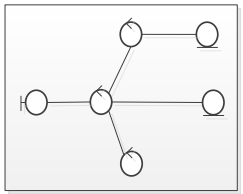
ユースケースモデル
／システムコンテキスト図

④ドメイン分割

Aドメイン

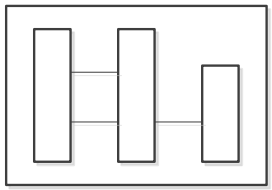
Dドメイン

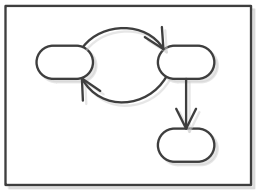
⑤ユースケースの振る舞い定義



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

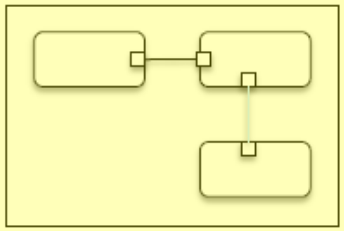




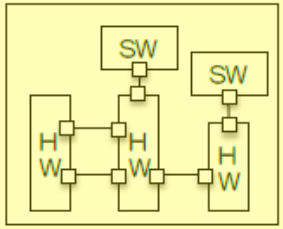
ブロック定義図　状態遷移図

等

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

等

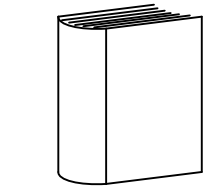
論理アーキテクチャ

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

基盤方式設計とは…これがアーキテクチャである。



基盤方式設計書

特に強い影響を受ける

要件

非機能要件

機能要件

＋客観的な尺度

品質属性

可用性

変更容易性

性能

使いやすさ

テスト容易性

セキュリティ

アーキテクチャ設計に要求される特性分類	基盤方式設計の記述内容	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	イベント処理			●			
	並行性とリソースアーキテクチャ			●			
	インスタンス管理		●	●			
	リアルタイム制御			●			
	メモリ管理		●	●			
	状態遷移／決定表の実装		●				
	多国語対応		●				
	セキュリティ管理				●		
	障害管理（可用性）	●					
	…						
	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	●	●	●	●	●	●

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、下図のような経験的な品質特性間の関係(※)をトレードオフするしかない。これは、一方の要求満足度を高めると、他方が損なわれることを示している。ただし、逆が同じ傾向になるとは限らない。

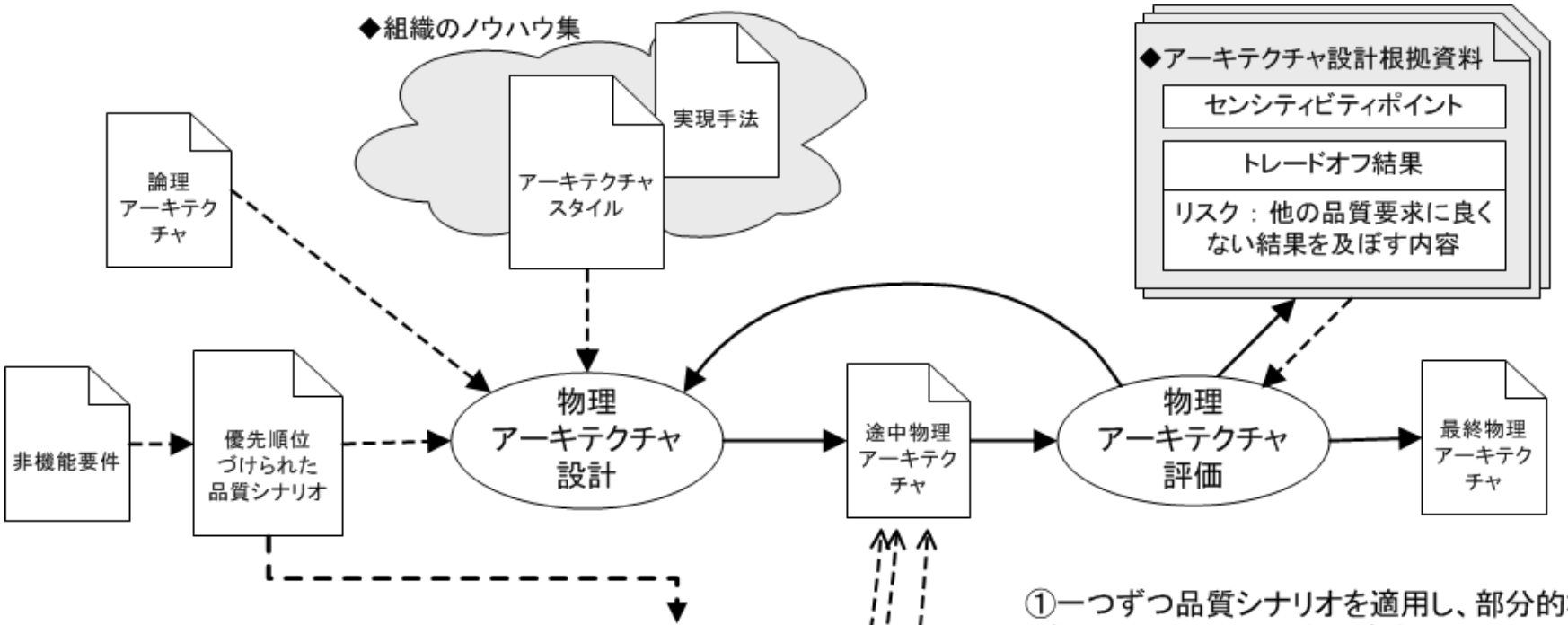
例)ソフトウェア構造の柔軟性を高めようとすると、性能は多少犠牲にせざるを得ない。

	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



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

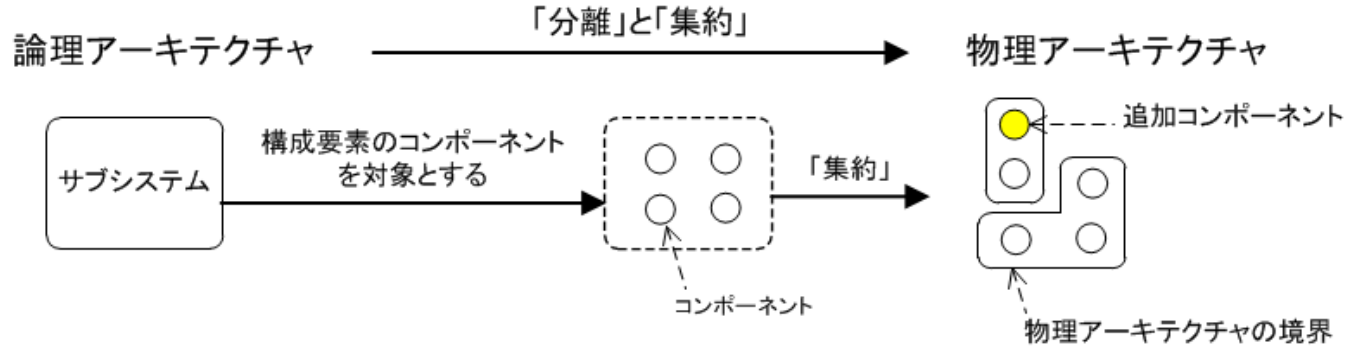
非機能要件の実装検討項目

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	●				●	●

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

MITSUBISHI SPACE SOFTWARE

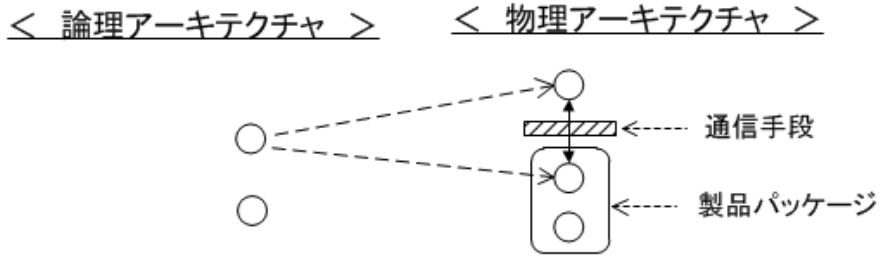
論理サブシステムから物理サブシステムへのマッピングは、通常1対1であるが、別のサブシステム枠になることもある。品質属性を考慮しながら、論理コンポーネントを、物理コンポーネントにマッピングすることで、その枠組みが決まる。



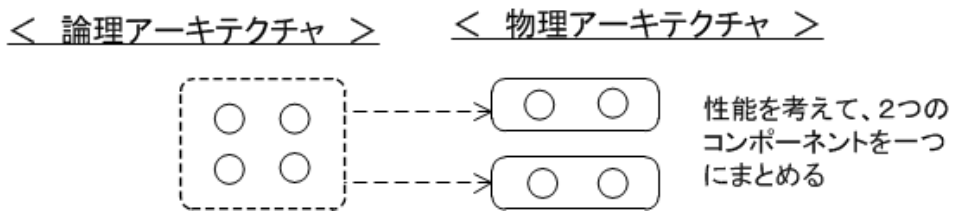
■＜論理アーキテクチャ＞　対　＜物理アーキテクチャ　＞　の関係

● ＜1＞　対　＜1＞　（　通常　）

● ＜1＞　対　＜多＞

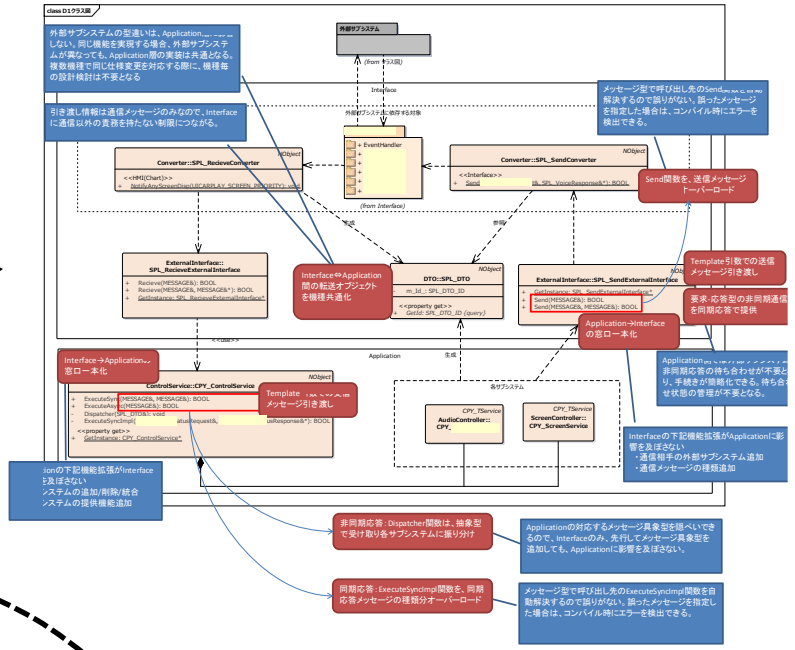


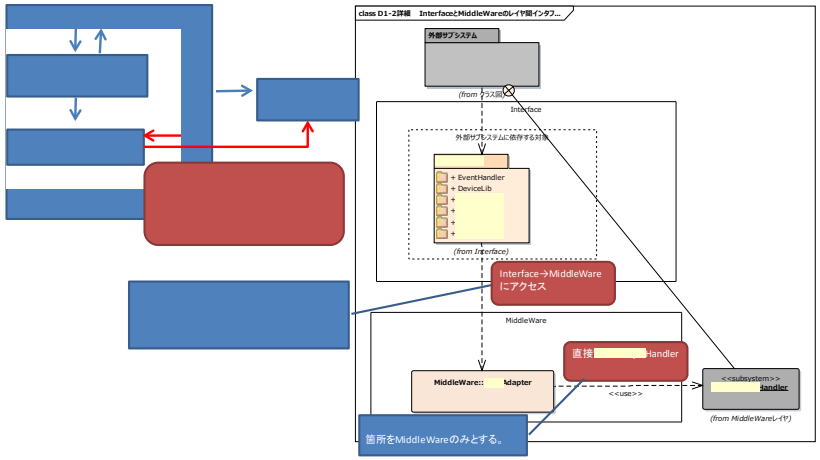
● ＜多＞　対　＜多＞



ATAM設計シートと設計過程のアーキテクチャ図

検討すべき非機能要件内容			詳細
解決すべき課題	（内容）		
前提事項	（内容）		
決定事項	（検討した最終アーキテクチャ）		
	分解課題	検討結果	詳細
		（トレードオフ&リスク）	
		（トレードオフ&リスク）	
		（トレードオフ&リスク）	





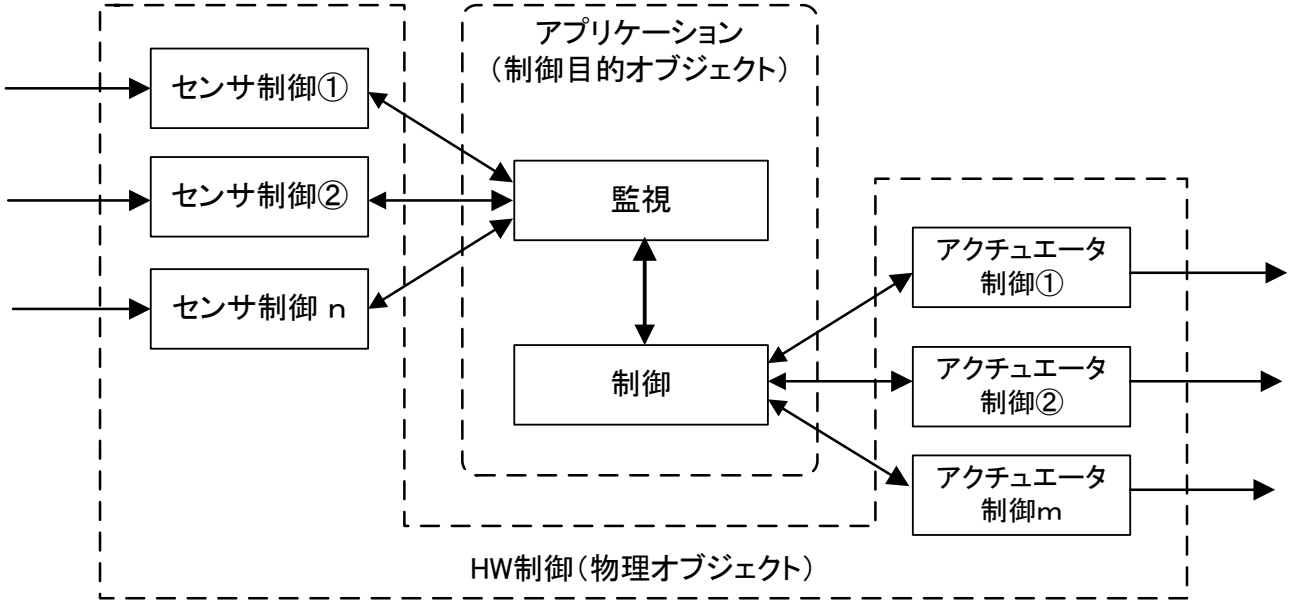
凡例）
○：メリット
×：デメリット

	案	作業量	可読性
メリット／デメリット	(a) Templateクラスを用いて量産を容易にする。	≪○優≫ typedefで型違いのメッセージを量産できる。 コーディング量が少ないので誤りも少なく済む。誤り修正の手間も少ない。	≪×≫ メッセージ定義ファイルが少なく一覧として確認しやすいが、変数名に意味のある名前を付けられないので、同じ型のメンバが複数になると、実装ミスが起きやすいデメリットがあるが、対策あり（欄外）
	(b) メッセージ仕様書から、コードを自動生成する。	≪×≫ 機種共通メッセージは、一度作成すると、複数の機種で流用できるクラスとなるので、自動生成に手間をかけても次に使う機会が無い。 また自動生成に足る設計書の記載は、ヘッダ定義作業と同等の負担となる。	≪○優≫ 生成するコードを統制できるので可読性が高くなる。
	(c) ヘッダファイルを手動で定義し、EAにリバースする。	≪○良≫ メッセージのインタフェースは、ヘッダに定義してビルドチェックを行うのが誤りを早く検出できるし、生産性も高い。 EAにはコードからリバースすればよい。	≪○良≫ EAのコード生成機能で枠のみではあるが統制されたソースを生成できる。 後は代入・初期化処理のみであり、統制すべき事項はコーディング規約で十分満たせる。
採用案	採用方針	可読性は必須とする。	
	採否	採用案	採用理由
		(c)採用	作業量も可読性も保てる為。

アーキテクチャ・スタイル蓄積が技術の見える化である。

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）

本開発手法による効果

手法適用の効果

■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、プロジェクトの成功確率が高まった（2007年以降、弊社基準のプロジェクト崩れはゼロ）。
- ✓ 設計書の出来不出来判定がし易くなり、設計スキル評価がし易くなった。
- ✓ 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）ツールやサブシステム定義ツール等、手法の支援ツールの整備が加速された（現在進行形：手順を固めることで、自動化がより見えてきた）

（※１）SPI Japan2016（デンソー）プロセス分析に基づくドキュメント再構成によるプロセスの改善

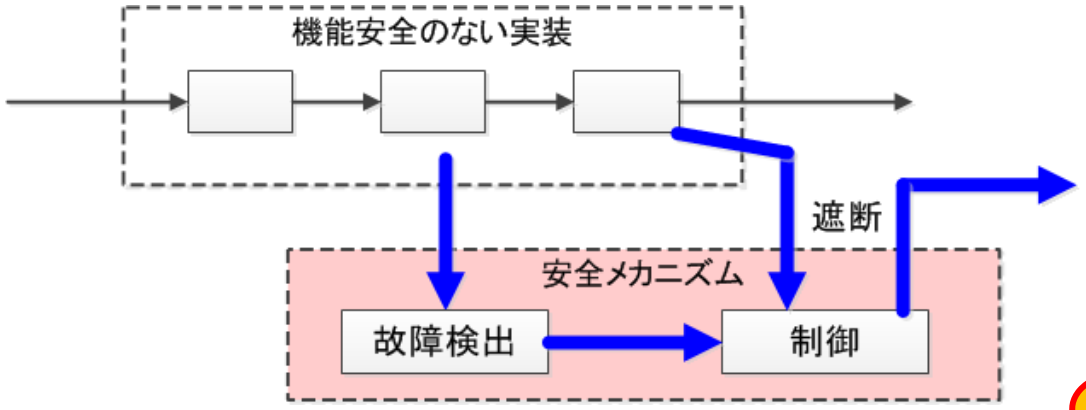
さらなる応用

機能安全への拡張

機能安全とは、

■機能安全とは『機能により電気電子システムの安全性を確保する』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。



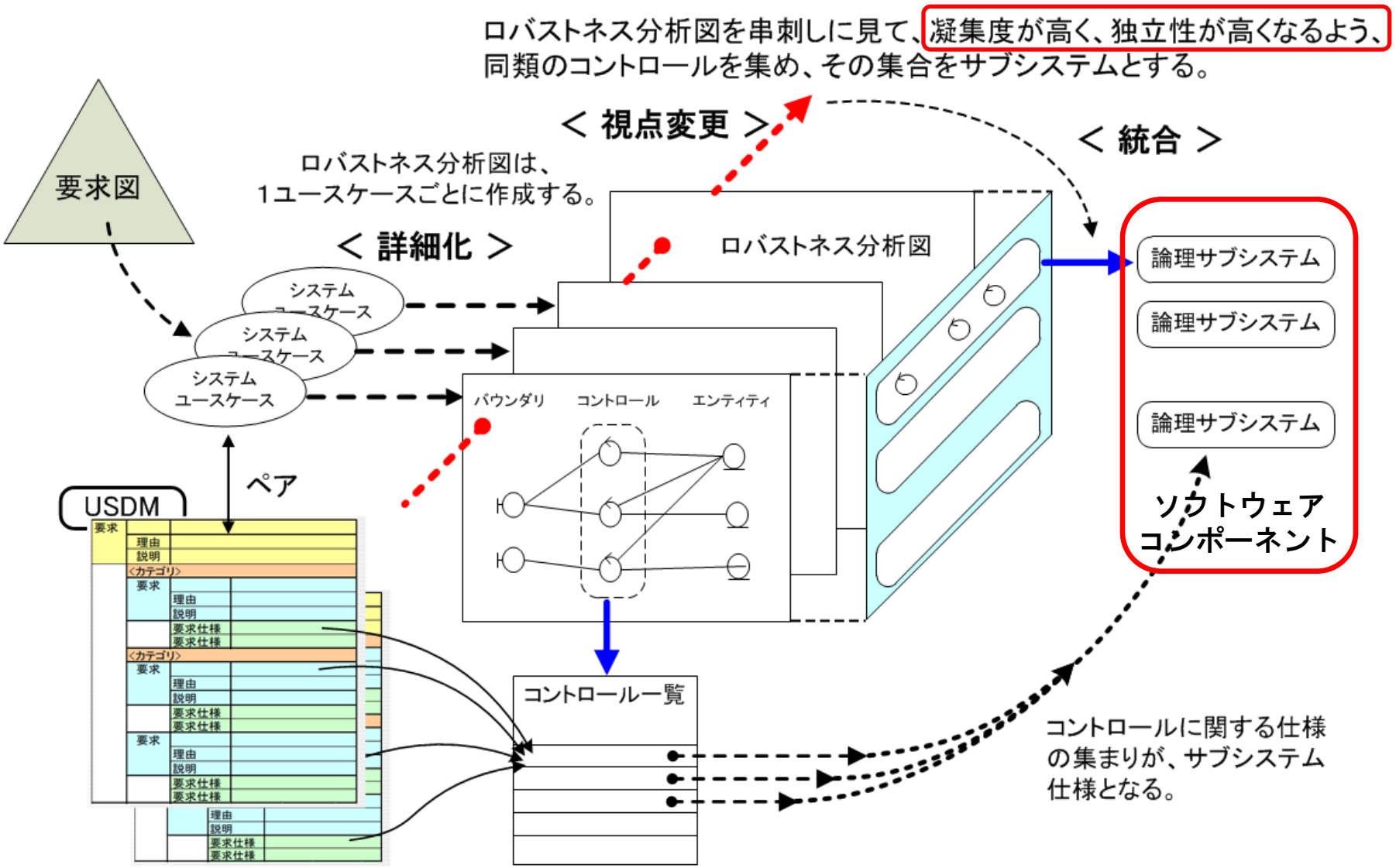
プロセス
マネージメント
だけではない！

■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。

サブシステム分割の基準（論理設計）

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。

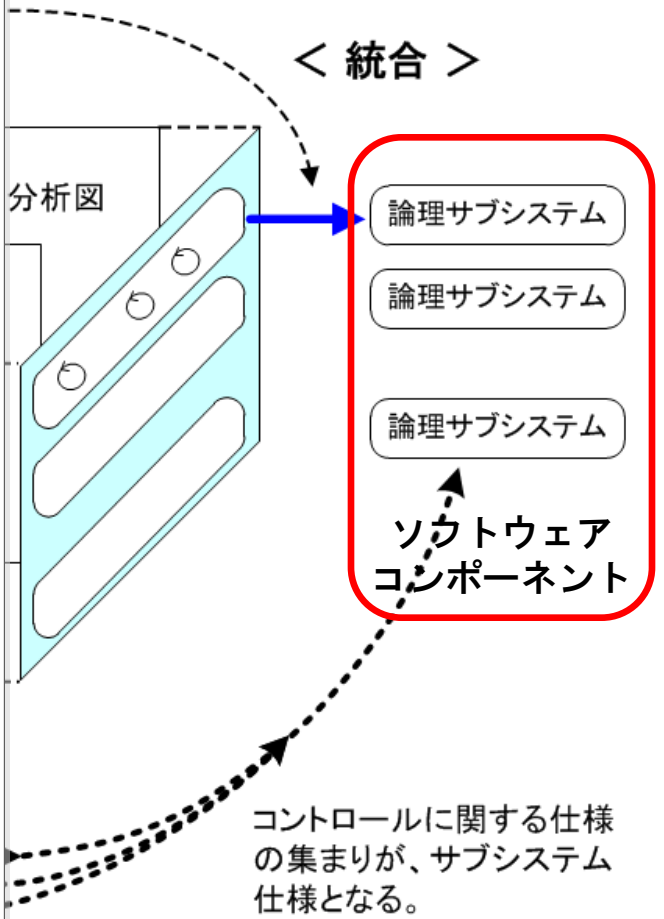


サブシステム分割の基準（論理設計）

コンポーネントの決め方は、既に説明したとおり（下図）。

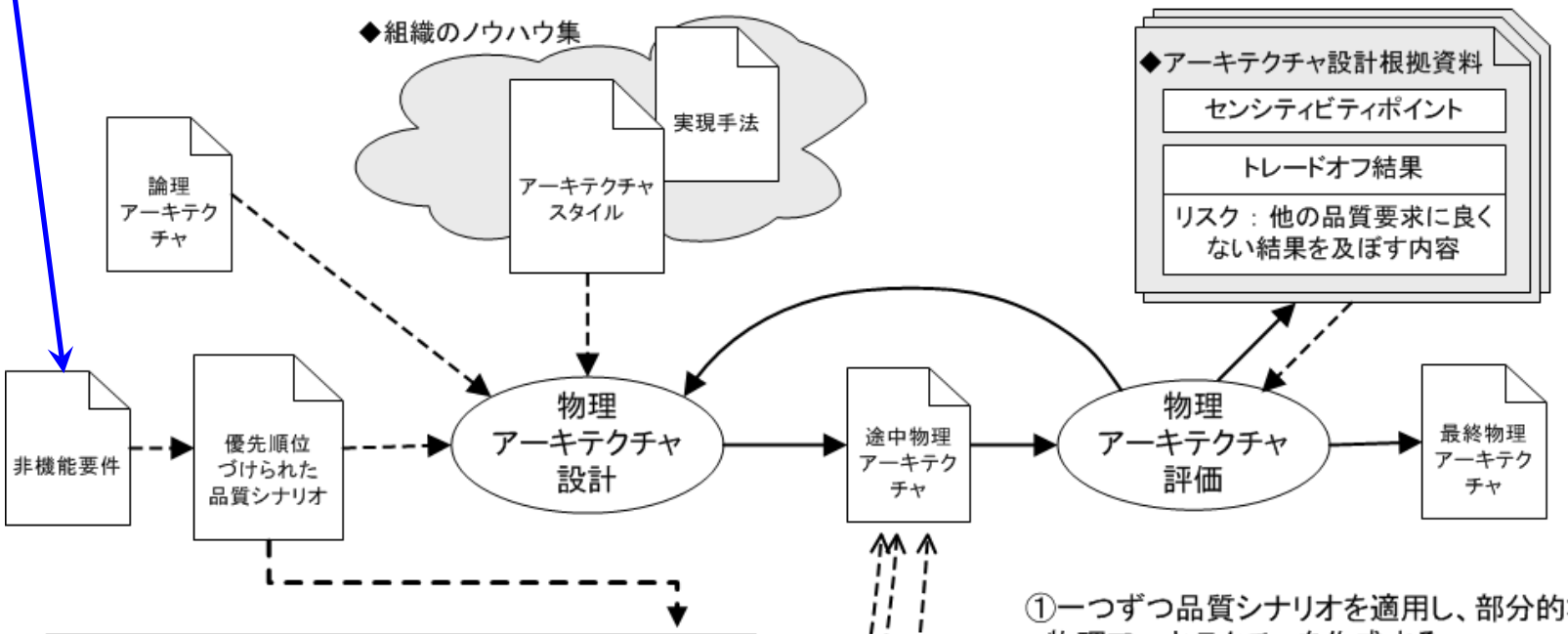
示せ」、と言っている。

見て、凝集度が高く、独立性が高くなるよう、
の集合をサブシステムとする。



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



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	●				●	●

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

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、
全手法の詳細を、ご説明させて頂きます。

ご清聴ありがとうございました。