FUJITSU
shaping tomorrow with you

JEITA組込み系ソフトウェア・ワークショップ2012
日本の組込み系開発におけるアーキテクト
～ アーキテクトは何を解決するか ～

アーキテクトが解決すること

2012年11月7日

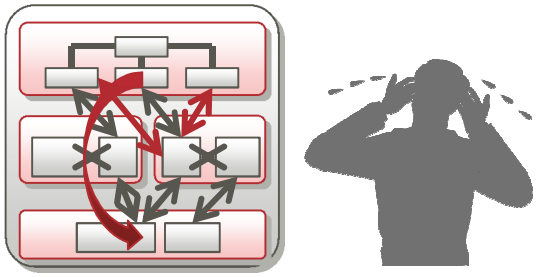
富士通株式会社　ネットワークビジネスグループ
共通開発本部　第一ソフトウェア開発統括部
シニア・プロフェッショナル・エンジニア　保土原　行彦

本日の内容

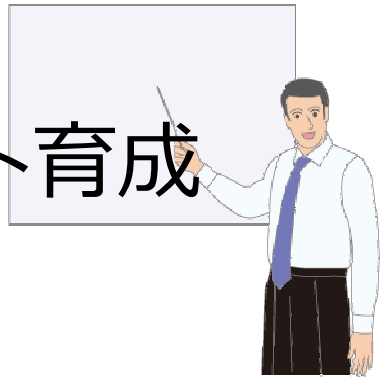
FUJITSU

- **アーキテクチャとは**
  - 定義とその目的

- **事例紹介**
  - アーキテクチャが崩れた事例



- **改善取り組み事例**
  - 崩れ防止の取り組みとアーキテクト育成



アーキテクチャとは

- アーキテクトのイメージ
- 製品開発における現場の課題
- 分かるようで解らないアーキテクチャ
- アーキテクチャの目的と特性

アーキテクトのイメージ

FUJITSU

■ アーキテクチャのイメージ

構造　仕組み　方式　⇒　再利用性（変化への対応力）が良いもの

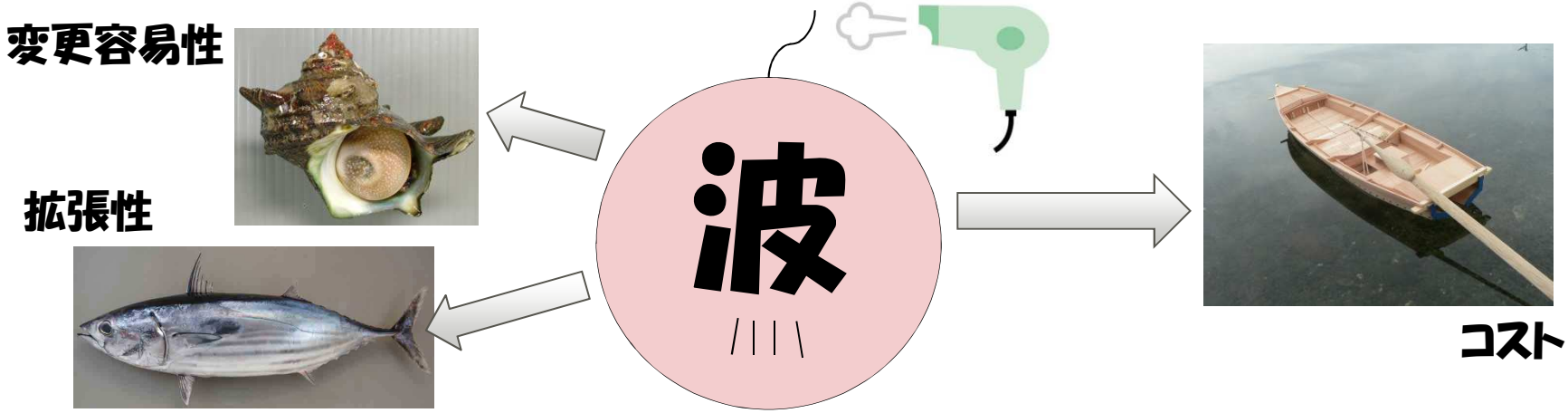
■ 変化への対応という意味でのアーキテクト

簡単には変えない頑固さ

頼れる大黒柱

アーキテクチャへのこだわり

目的を俯瞰した調整力



製品開発における現場の課題

FUJITSU

■ 開発現場における課題

派生開発

- ネットワーク機器におけるシリーズ品
- 機器の用途に応じた機能具備・配備

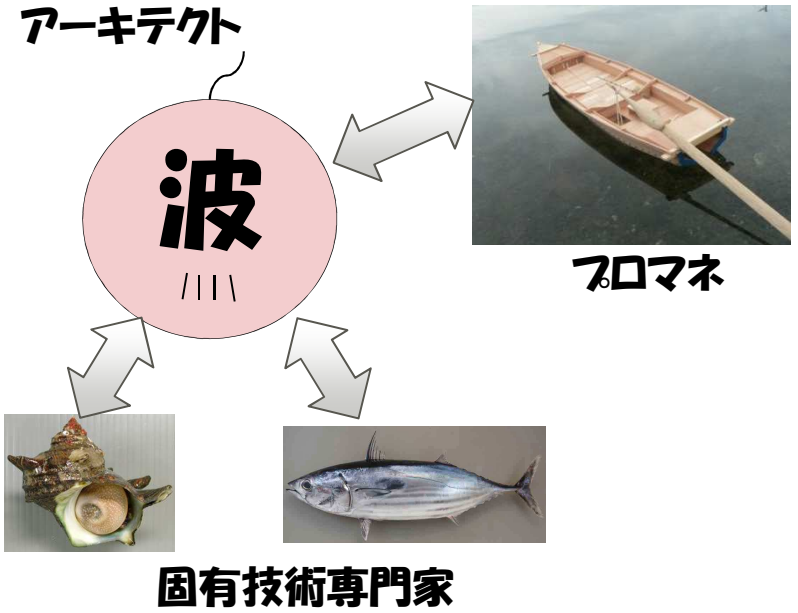
スケーリング

- 収容する対象によるスケーリング（リンク／パス／アソシエーション）

移行性

- デバイス統廃合を鑑みたコンポーネントインタフェースの実現

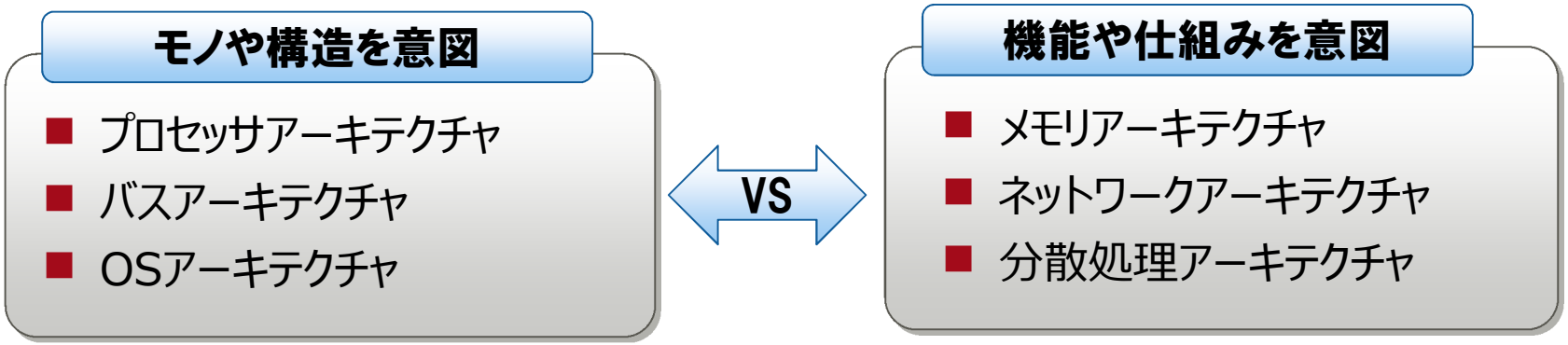
アーキテクチャで解決



分かるようで解らないアーキテクチャ

FUJITSU

■ 目的観点でのアーキテクチャの定義



アーキテクチャという言葉が意味するものが多種多様なため、
異なるステークホルダー間で議論しても分かり合えない

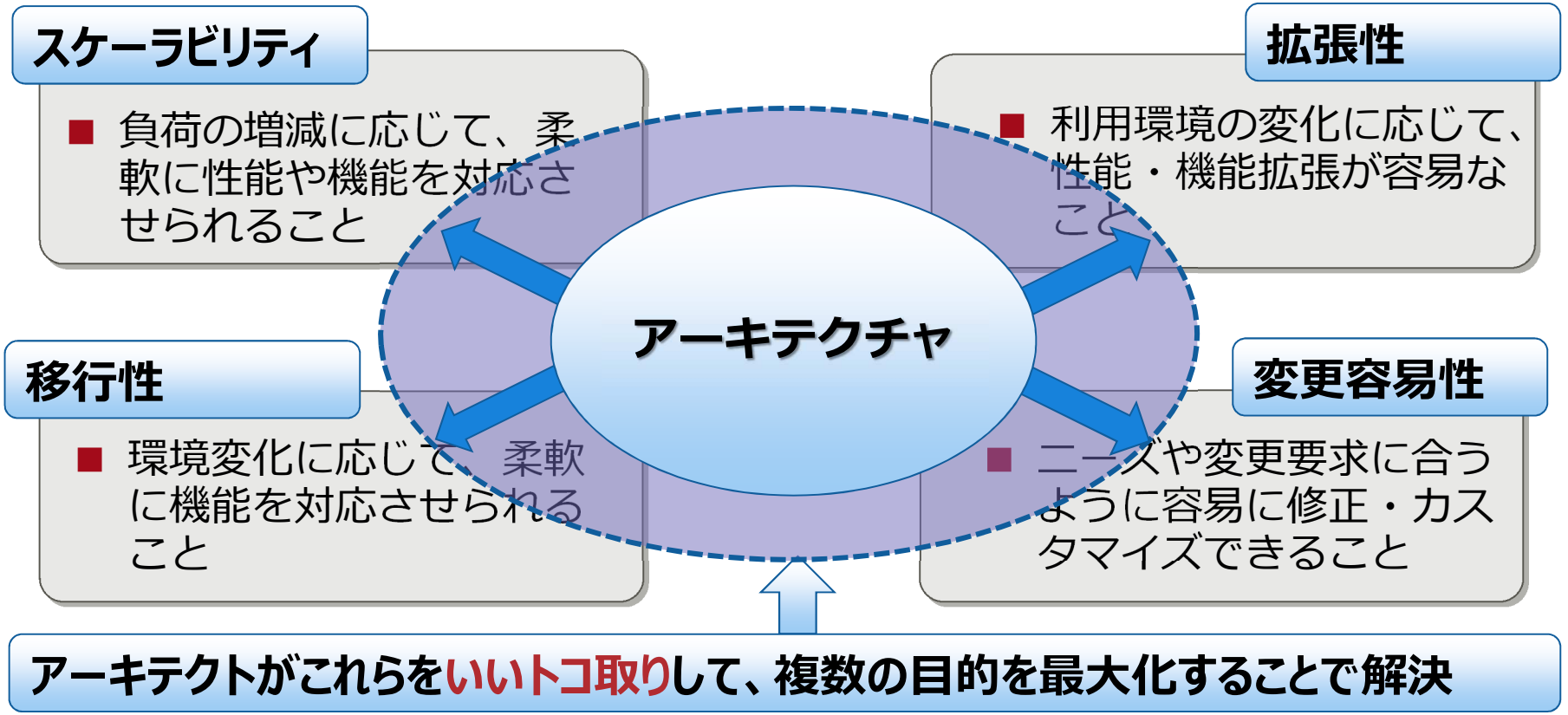
目的に主眼を置いた定義として

ソフトウェア開発の際に、将来変化を予測し、この変化に応じてQCDを最小限にする要素を予め設計等に仕込むこと（Q：Quality，C:Cost，D:Delivery）

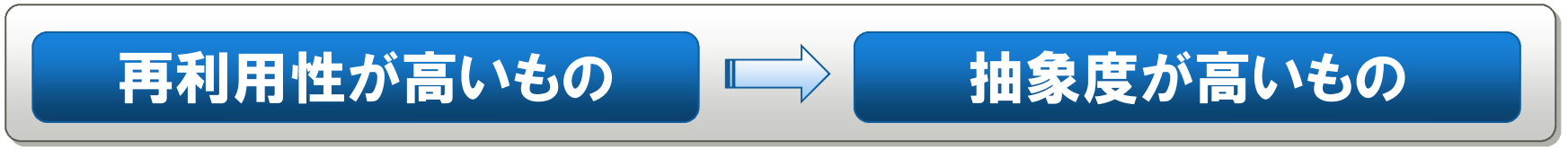
アーキテクチャの目的と特性

FUJITSU

■ アーキテクチャはQCDを解決するための手段



■ アーキテクチャの特性



FUJITSU

事例紹介

- アーキテクチャで規定しているもの
- アーキテクチャが崩れる時
- 【事例１】派生開発
- 【事例２】スケーリング
- アーキテクチャがなぜ崩れたか

アーキテクチャで規定しているもの

FUJITSU

■ 機能分割の結果生まれる外部と内部のうち、主に外部を定義

外部

- 部分間の相互作用と各部分の外的特性

アーキテクチャで規定しているもの

- **論理的構造／時間的構造**
- **静的構造／動的構造**

表現するもの	内　容
コンポーネント	レイヤ、プロセス、タスク、モジュール
コンポーネントの外的特性	共有リソース、エラー処理／特性、パフォーマンス特性

構造とインタフェース（実装方法は未定義）

内部

- 基本設計
- 詳細設計

アーキテクチャでは規定していない

コンポーネント内部を規定しないことで抽象度を高める

アーキテクチャが崩れる時

FUJITSU

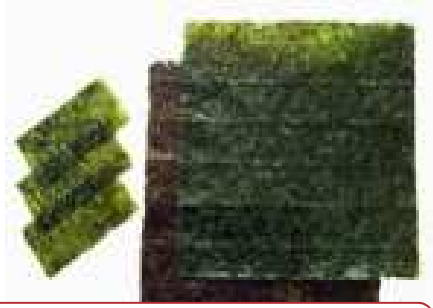
- **派生開発の繰り返し**
  - 構造の劣化による再利用率の低下
  - インタフェース変更による再利用率の低下

派生開発　⇒　事例１

- **ハードウェア改変等でのスケール未達**
  - ハード・ソフトインタフェース変更による影響
  - プロセッサやOSアーキテクチャ変化による影響

スケーリング　⇒　事例２

- **大がかりな機能追加**
  - 元のアーキテクチャ構想が解らずに機能追加する
  - そもそも基礎工事のやり直しが必要だった？

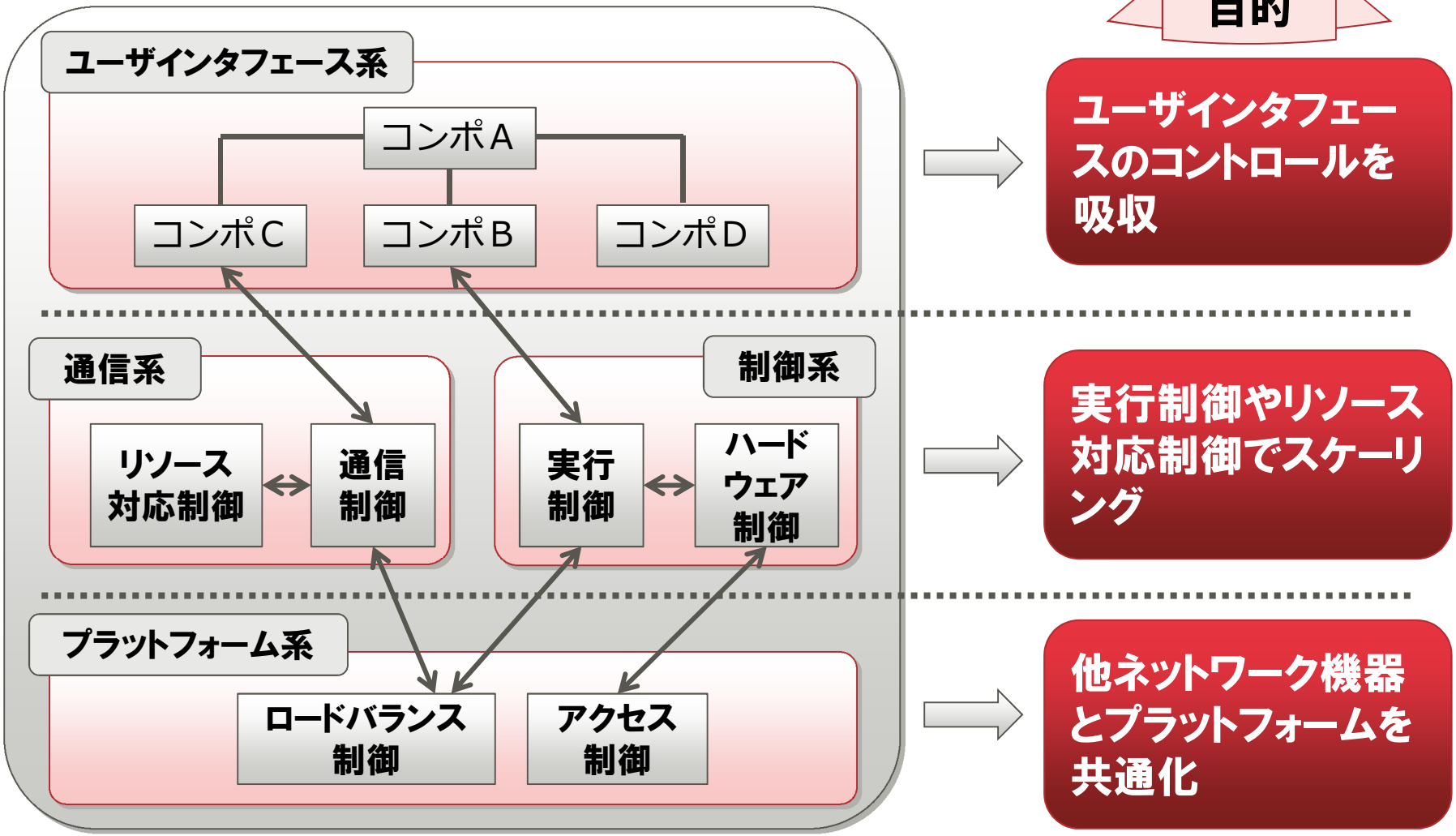


元の構想の理解不足や、今後の変化の考慮無しに変更している

【事例１】 派生開発 ～アーキテクチャの目的～

FUJITSU

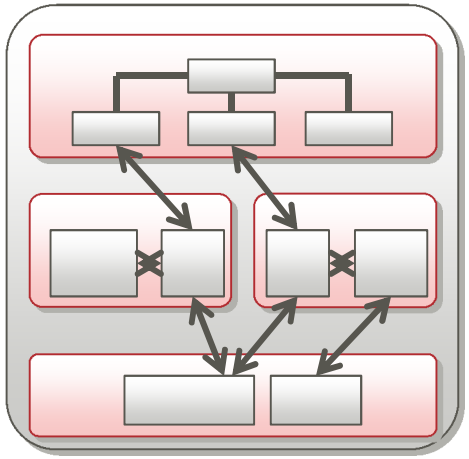
■ **大規模開発をサブシステム分割、シリーズ製品対応をサブシステム内で閉じようとした**

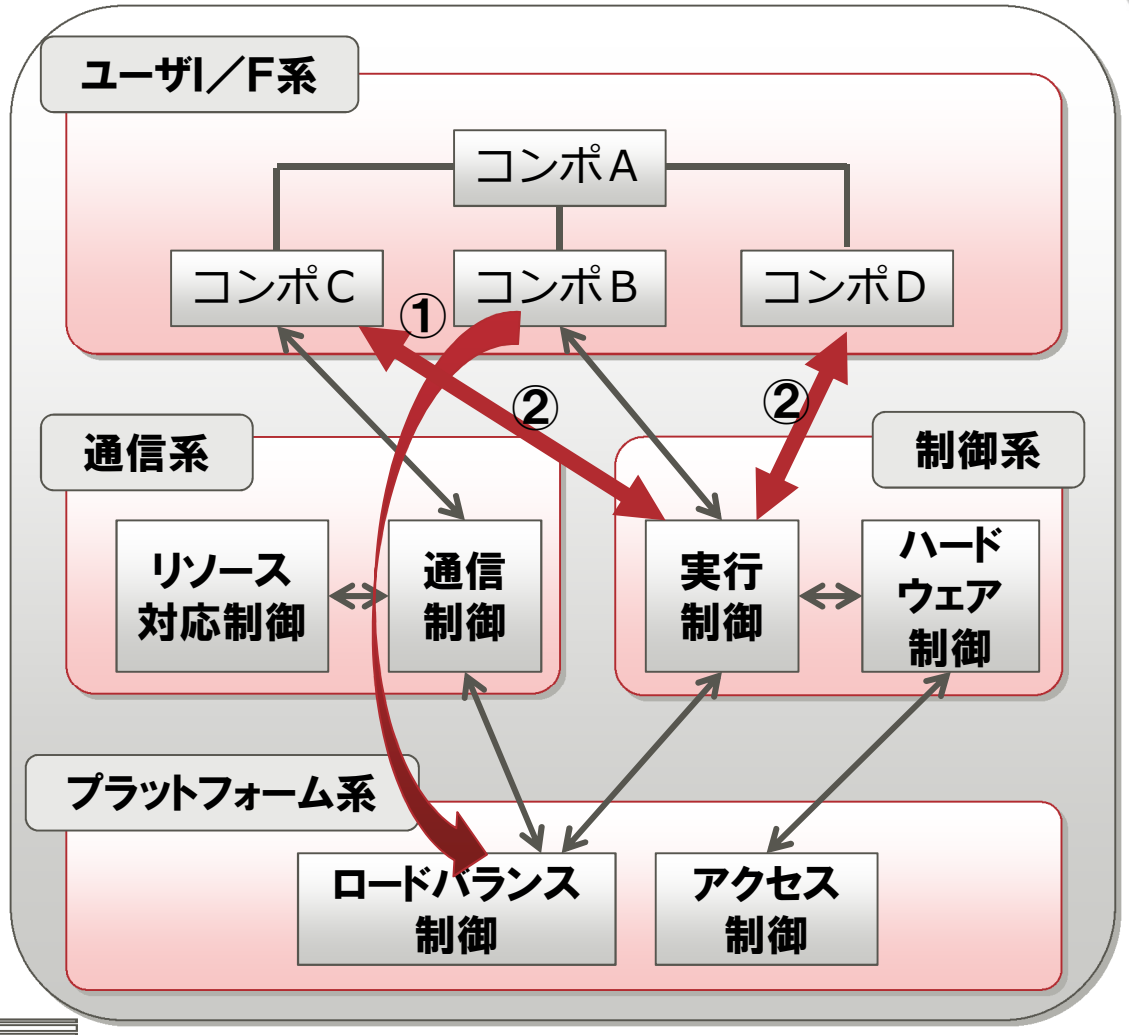


【事例１】 派生開発　～変更によるアーキ崩れ～

FUJITSU

■ 再利用率の低下





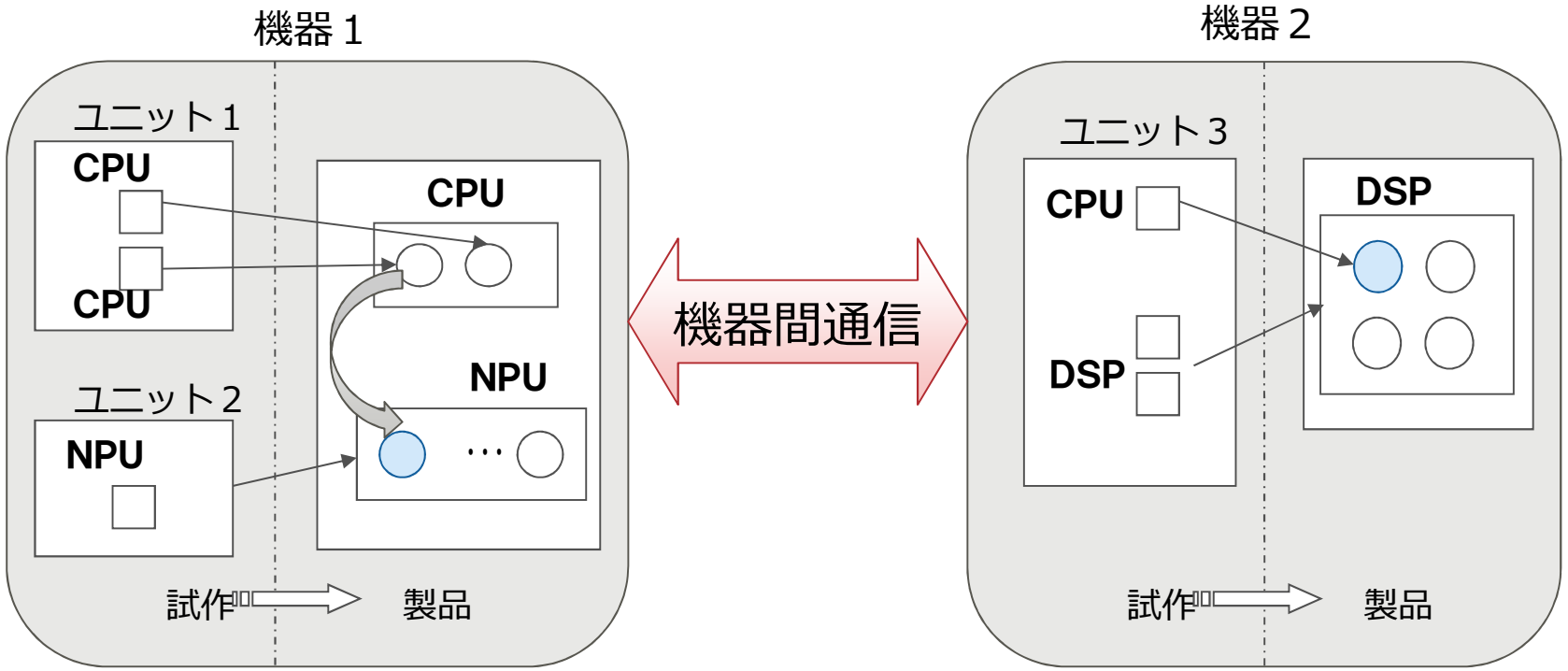
- ①：階層越えのインタフェース
  - レスポンス向上のためロードバランサに対して直接制御
  - 次の派生開発でレスポンス低下
- ②：依存関係の増加
  - コンポC，Dからの実行制御と、上位へのデータ依存

「コンポB＋実行制御＋ロードバランサ」の組み合わせによる再利用量が減少

【事例２】スケーラビリティ　～アーキテクチャの目的～

FUJITSU

- **プロセッサやOS変更を睨み、データ通信処理をスケーラブルにしようとした**



- **CPU機能を１チップに統合**
- **CPU機能の一部をNPUにオフロード**
- **ユニット３CPUはOS変更によるインタフェース変更あり**
- **ユニット３からユニット１への機能配備変更（試作時計画あり）**

【事例２】スケーラビリティ ～変更によるアーキ崩れ～

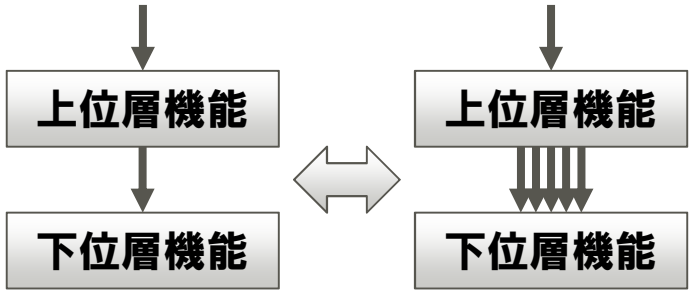
FUJITSU

- **スケーラブルで無くなった**
  - **機器間、ユニット間の通信層とのインタフェース（通信方式）を変更**
    - メッセージパッシングがオーバヘッドと担当が問題視、共有メモリ方式でバグ対処した
    - 設計当初の目論見とは異なる共有メモリ方式のため、データレースペナルティによりスケールしなくなった

- **再利用性が落ちた**
  - **OS統合に備えていたAPIインタフェースを変更**
    - インタフェース部のオーバヘッド削減のための変更
    - OSのメモリモデル（仮想アドレッシング／リアルアドレッシング）差分を吸収するインタフェースの変更

※かなり大雑把に一般化すると

複数処理を下位層に実行させる時のインタフェース変更（単発／複数）



アーキテクチャがなぜ崩れたか

FUJITSU

ソフト改変したソフトウェアエンジニアによる振り返り

- **再利用構想が解らなかった**
  - 派生開発における再利用対象のコンポーネントが設計書で読み取れなかった
  - OS統合の目論見がある中で、候補OS毎のインタフェース規定点のメリットなどが解らなかった
  - 変更時の禁則事例が示されていると良いと思った

アーキテクチャ**目的・構想**

- **インタフェースの実現手段の目論見が解らなかった**
  - 将来採用予定のプロセッサ、OSにおいてメッセージパッシング方式の方が有利であることを理解できていなかった

インタフェース**手段**

- **実行制御の内部の仕組みを理解できていなかった**
  - 仕組みによるメリット
  - 改変による影響、デメリット

コンポ内部の**仕組み**

アーキテクチャ取り組み事例

- アーキテクチャが崩れる要因
- アーキテクチャ崩れ防止への取り組み
- ドキュメント強化内容
- 崩れ防止のためのフォーメーション
- アーキテクト育成の３つの方法

アーキテクチャが崩れる要因

FUJITSU

アーキテクトの意図（目的・構想）が伝わらない

- **親の心、子知らず**
- **その心が伝わるような設計書でない、または伝えてない**

目論見を意識せずコンポーネントインタフェースを変更

- **場当たり的な変更で、インタフェースを変えてしまう**
- **特にバグ改修や、処理性能等の改善によるプログラム改変時**

コンポーネント内部の仕組みの理解不足

- **アーキテクチャは抽象度を高くするため、コンポーネント内部までは言及しない**
- **設計の都合で疎結合だったものが、データ関連性で密になる**

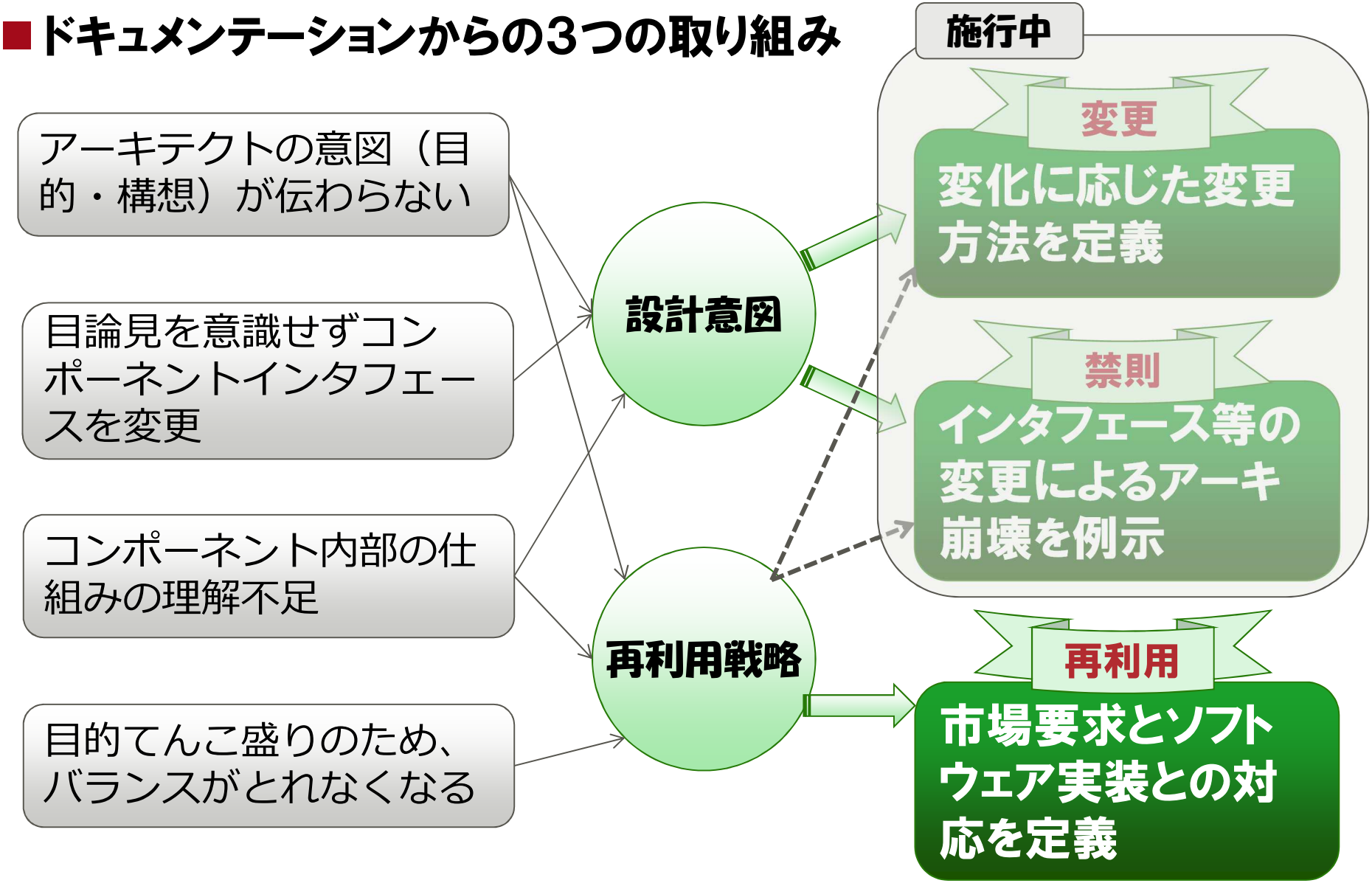
目的てんこ盛りのため、バランスがとれなくなる

- **スケーラビリティ　vs　移行性**

アーキテクチャ崩れ防止への取り組み

FUJITSU

■ドキュメンテーションからの3つの取り組み

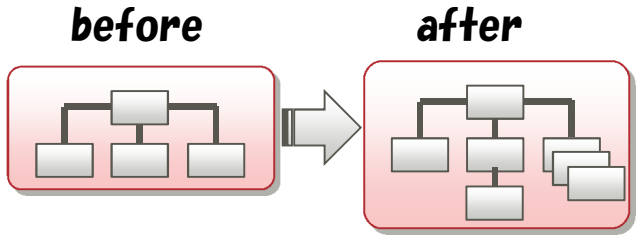


ドキュメント強化内容

FUJITSU

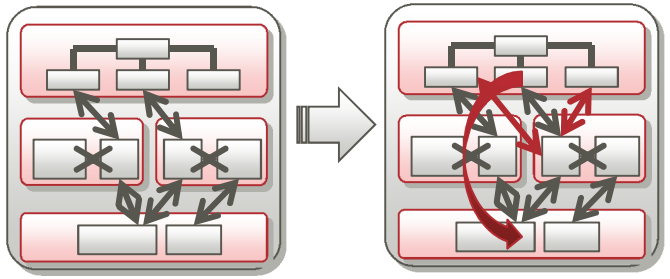
変更　変化に応じた変更方法を伝える

- **アーキテクチャの目的・意図を示す**
- **変動要素毎に、変更方法を定義する**



禁則　崩れる例を伝える

- **採用してはならない変更方法を定義する**
- **デメリットも合せて示す**



再利用　変化に応じた再利用対象を伝える

- **MFL（Marketing Feature List）**

市場要求			重要度			ソフトウェア関連機能	
概要	詳細1	詳細2	お客様要求	ビジネス目標	実現難易度	機能名称	固定／変動
低コスト化	開発費削減		◎	◎	△		固定
	部品点数削減		◎	◎	○	ｘｘドライバ	変動

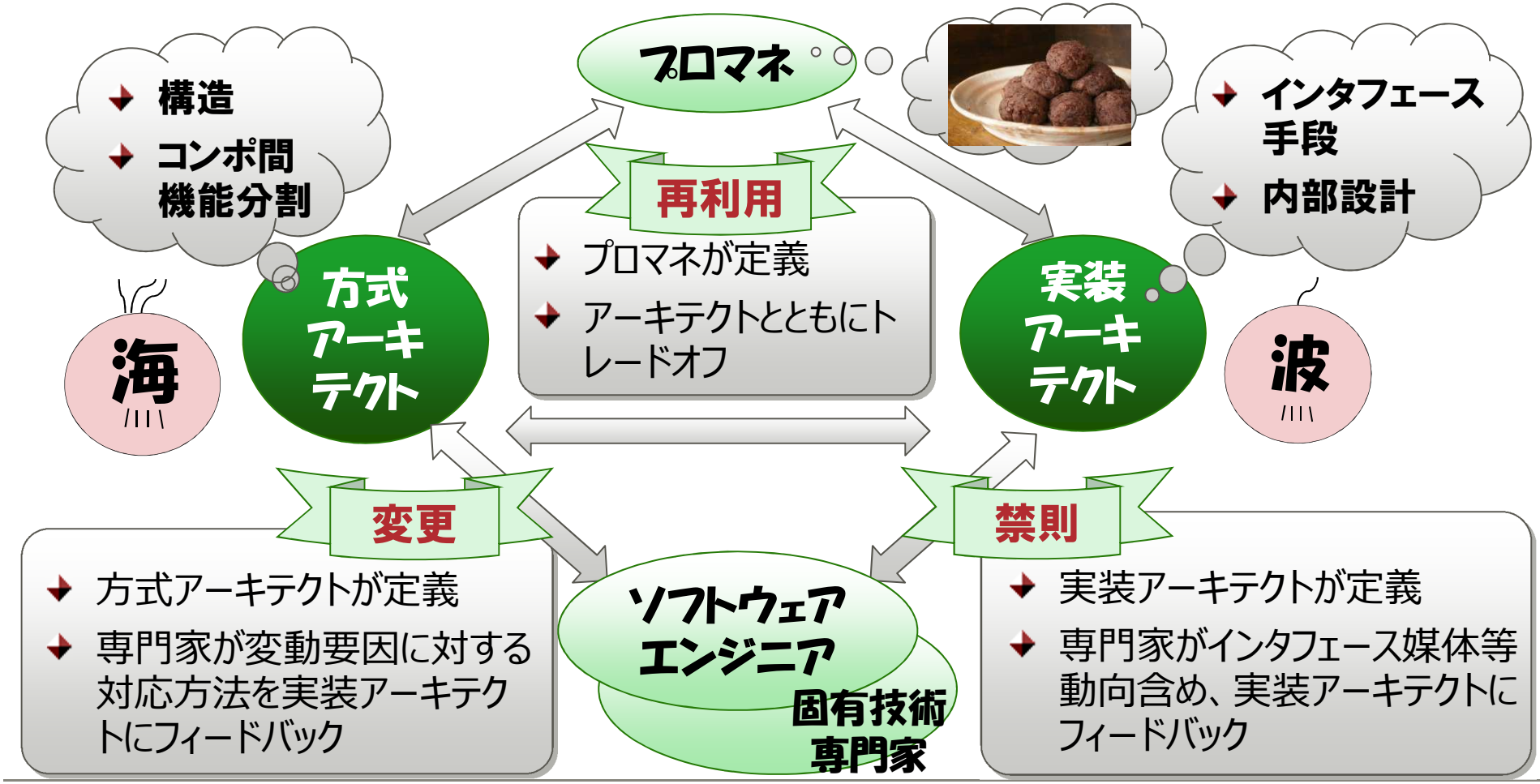
詳細2にて分割した機能等を定義、再利用可能なものを抽出

固定部の構造やインタフェースと、変動部の変動要素を定義

崩れ防止のためのフォーメーション

FUJITSU

- ソフトウェアアーキテクトの役割を2分割（大規模の場合）
- 崩壊抑止要因をステークホルダー間合意事項にマッピング
- 専門家およびエンジニアが必要に応じアーキテクトにフィードバック



アーキテクト育成の3つの方法

FUJITSU

欧米との決定的な違いがある「品質を心配する意識」を日本の強みと認識し、これをベースに育成（人的擦り合わせ）

- 上級アーキテクトによる育成
  - **いいトコ取り(トレードオフ)の技術を伝承**
  - **機能分割の技術を伝承**

本質は**分割**にあり！

経験に基づく指導

- 固有技術の専門家による育成
  - **設計勘所の観点**
  - **固有技術の将来動向**

勘所は専門家が**フォロー**！

設計や品質の勘所をインプット

- プロマネによるサポート
  - **最後まで面倒を見させる**
  - **ソフトウェアの再利用戦略を教える**

経験も重要！

経験させる場を提供

まとめ

FUJITSU

■ アーキテクトが解決すること

QCDを解決するために、再利用性を高めるように仕込む（機能分割がキモ）

継続的目的達成のためにアーキテクチャをキープ、インタフェースを頑固に守る

■ アーキテクチャを崩さないようにするために

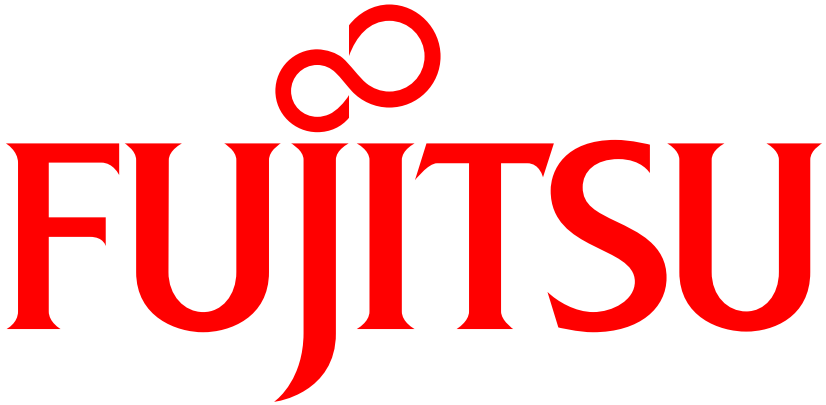
変化に応じた変更方法を予め定義して設計書に織り込む

市場要求とプラットフォーム変化を含めたソフトウェア実装対応を定義

■ アーキテクトの育成

周りも育てる（プロマネ／上級アーキテクト／固有技術専門家）

ご清聴ありがとうございました



shaping tomorrow with you