2025/10/22 SPI Japan

アーキテクチャ進化の意思決定におけるメトリクスの活用

過去発表事例の振り返りと新たなメトリクス活用の可能性

TechMatrix

テクマトリックス株式会社
ソフトウェアエンジニアリング事業部
牧 隆史

本日の内容

I. 過去発表事例の概要紹介

II. アーキテクチャの位置づけの変化

III.考察

Ⅰ.　過去発表事例の概要紹介

背景

- コンシューマ商品のプロダクトライン開発
- 機能追加をトリガーとした実装の崩壊
- アーキテクチャ見直しの必要性

  ⇒属人的でない方法の模索
- 論文
  - 牧 隆史、岸 知二「プロダクトライン開発におけるアーキテクチャリファクタリングの意思決定法」、情報処理学会論文誌 Vol.55 No.2 pp.1069-1078 (Feb. 2014)

モチベーション

PLDは長期にわたるため、新たな顧客要求対応のためアーキテクチャのリファクタリングが必要となる。

しかしながら、

製品開発と並行してアーキテクチャリファクタリングを行う場合、課題が複数あるため、リファクタリングの実施判断が難しく、これまでその手法がなかった。

- 課題
  - 実装アーキテクチャ自体の問題
  - 参照アーキテクチャ自体の問題
  - 両社の乖離の問題

これを解決するため、

問題を総合的に判断した上でリファクタリングの優先度を判断する手法を提案したい。

当時の既存技術

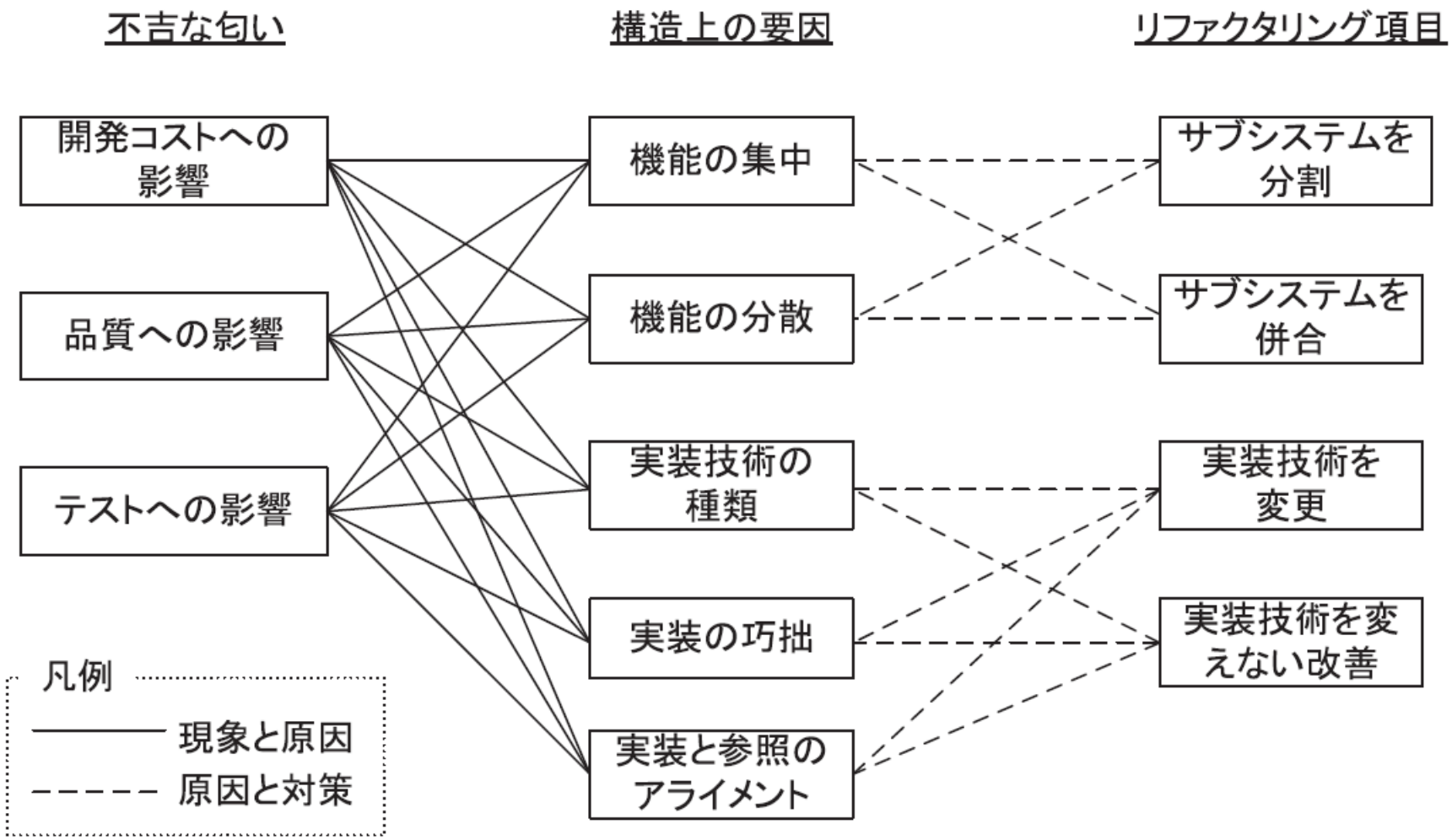
- アーキテクチャ評価手法
  - ATAM, SAAM, PuLSE-DSSA
- ソースコードリファクタリングの技法
  - Fowler, Kerievsky, Roock

⇒いずれも個別の手法及び技法であり、実装を踏まえてリファクタリングの意思決定をする手法ではない

提案した手法の骨子

- プロジェクトでの気づき（bad smells）を出発点としてリファクタリングニーズを見つける
- 原因を分析し、問題の大きさを参照と実装アーキテクチャ上の問題に分けて定量化する
- 定量化した結果のポートフォリオ分析をリファクタリングの意思決定に用いる

基本的な概念間の関係(例)



構造上の要因の定量化

構造上の要因	問題の在処	計測対象の例	5段階評価指標 小 ←（問題の大きさ）→ 大				
			1	2	3	4	5
サブシステムの役割定義が広すぎる	参照	サブシステムのソースコード行数(LOC)	<(L1)	<(L2)	<(L3)	<(L4)	(L4)≦
全体から参照されるものが多い	参照	依存関係の数	<(N1)	<(N2)	<(N3)	<(N4)	(N4)≦
サブシステム間の依存関係が多い	参照	依存関係の数	<(N1)	<(N2)	<(N3)	<(N4)	(N4)≦
コンパイルスイッチが多い	実装	コンパイルスイッチの出現密度	<(R1)%	<(R2)%	<(R3)%	<(R4)%	(R4)%≦
関数が長い	実装	関数の平均行数	<(L1)	<(L2)	<(L3)	<(L4)	(L4)≦
複雑度が高い	実装	サイクロマチック複雑度の平均値	<(C1)	<(C2)	<(C3)	<(C4)	(C4)≦
結合度が高い	実装	依存関係の数	<(N1)	<(N2)	<(N3)	<(N4)	(N4)≦
サブシステムの構成要素が異なる	実装	機能差分の数	<(F1)	<(F2)	<(F3)	<(F4)	(F4)≦
依存関係の齟齬	実装	依存関係の数	<(N1)	<(N2)	<(N3)	<(N4)	(N4)≦

「不吉な匂い」の例

- 機能追加時に影響範囲を確認すべき箇所が多く工数がかかっている(S1)
- 他のサブシステムと比較して不具合多発の傾向がみられる (S2)
- テストの組み合わせが多い (S3)

複数の機種開発において、繰り返し見られる現象

「構造上の要因」の例

参照アーキテクチャに起因する構造上の要因

- 特定サブシステムの役割定義が広すぎ、特定サブシステムにコードが偏在している(Pr1)
- サブシステム間の依存関係が多い(Pr2)

実装アーキテクチャに起因する構造上の要因

- コンパイルスイッチが多いために保守性が低い(Pi1)
- 個々の関数が長い、または複雑度が高い(Pi2)
- 依存関係の齟齬が存在する(Pi3)

「リファクタリング項目」の例

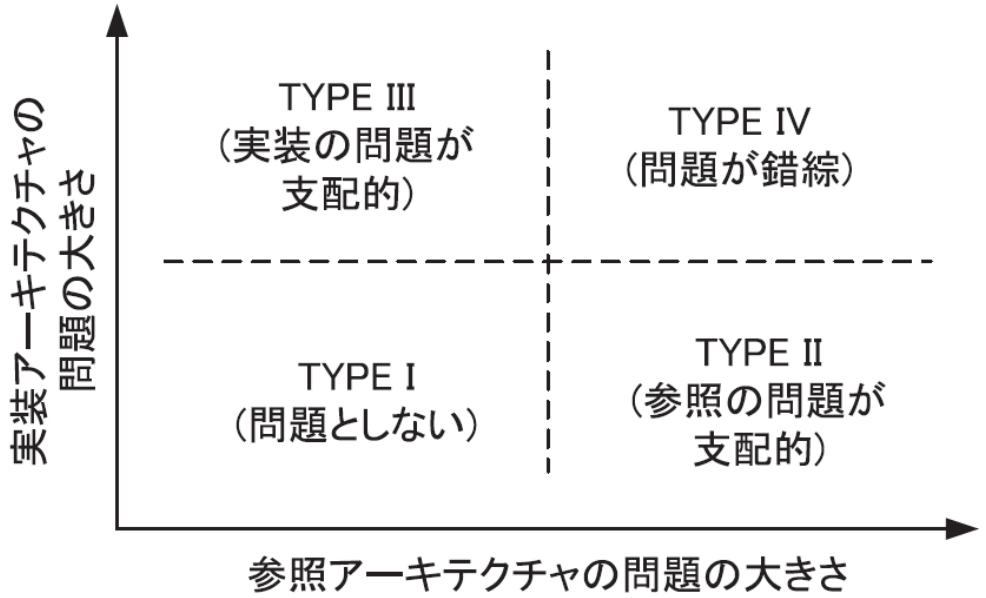
参照アーキテクチャに対するリファクタリング項目

- 特定の関心事に基づく処理を別ドメインに移動する(Rr1)
- サブシステムを新規に作成、或いは他と併合する(Rr2)
- サブシステムの役割を変更する(Rr3)

実装アーキテクチャに対するリファクタリング項目

- 変動点の実現方法をコンパイルスイッチから別の方式に変更する(Ri1)
- サブシステム間で処理を移設する (Ri2)

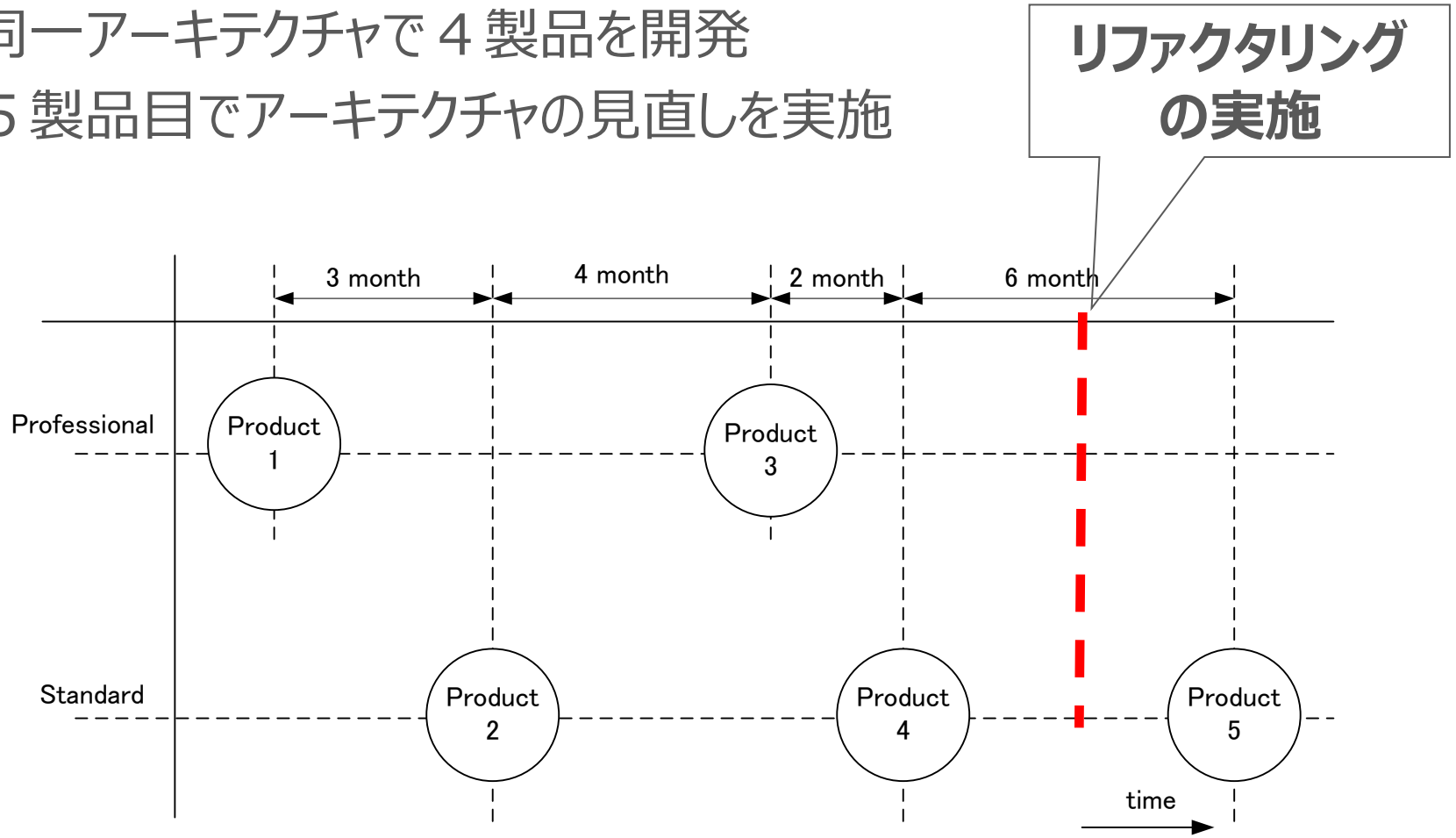
ポートフォリオ分析(タイプ別の特性傾向)



- TYPE I – 問題としない：他のTYPEと比較して実装上も参照上も問題が小さい場合である。リファクタリングの優先度は他の区分よりも低いと判断できる。
- TYPE II – 参照の問題が支配的：相対的に参照アーキテクチャ問題が大きい場合である。実装アーキテクチャの問題は少ないため、実装は参照アーキテクチャに忠実に行われているが、参照アーキテクチャ自体が陳腐化して変更要求に耐えられなくなってきている状況と考えられる。
- TYPE III – 実装の問題が支配的：実装アーキテクチャの問題が相対的に大きい場合である。参照アーキテクチャの問題は大きくなく、実装が参照から乖離しているケースがこれに含まれる。実装の修正により問題は解消することが多い。
- TYPE IV – 問題が錯綜：実装も参照も問題が大きい場合である。参照アーキテクチャをこのままにしいて実装だけ修正しても参照アーキテクチャの問題が残るので、解決するにはまず参照アーキテクチャの修正を行うべきである。

プロジェクトでのリファクタリング(1)

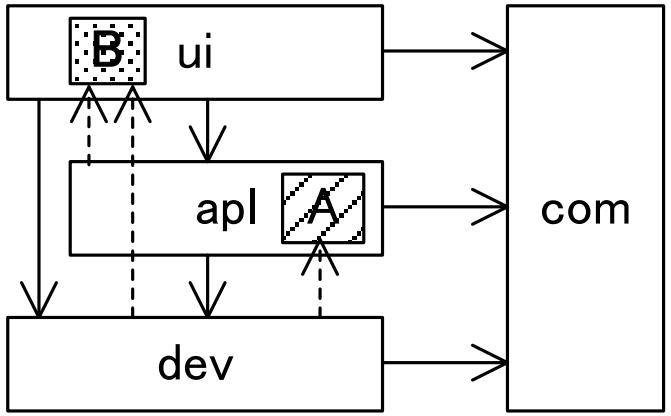
- アーキテクチャ変更の概要
  - 同一アーキテクチャで４製品を開発
  - ５製品目でアーキテクチャの見直しを実施



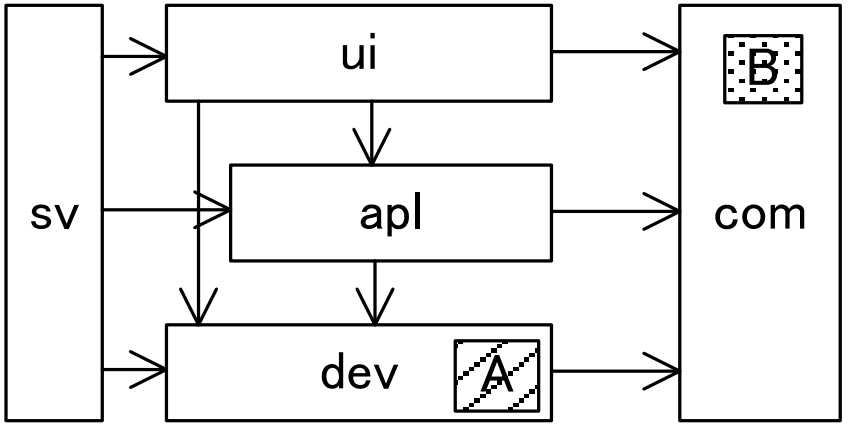
プロジェクトでのリファクタリング(2)

アーキテクチャ変更の概要

- 主に依存関係の齟齬解消を中心として専門家が判断



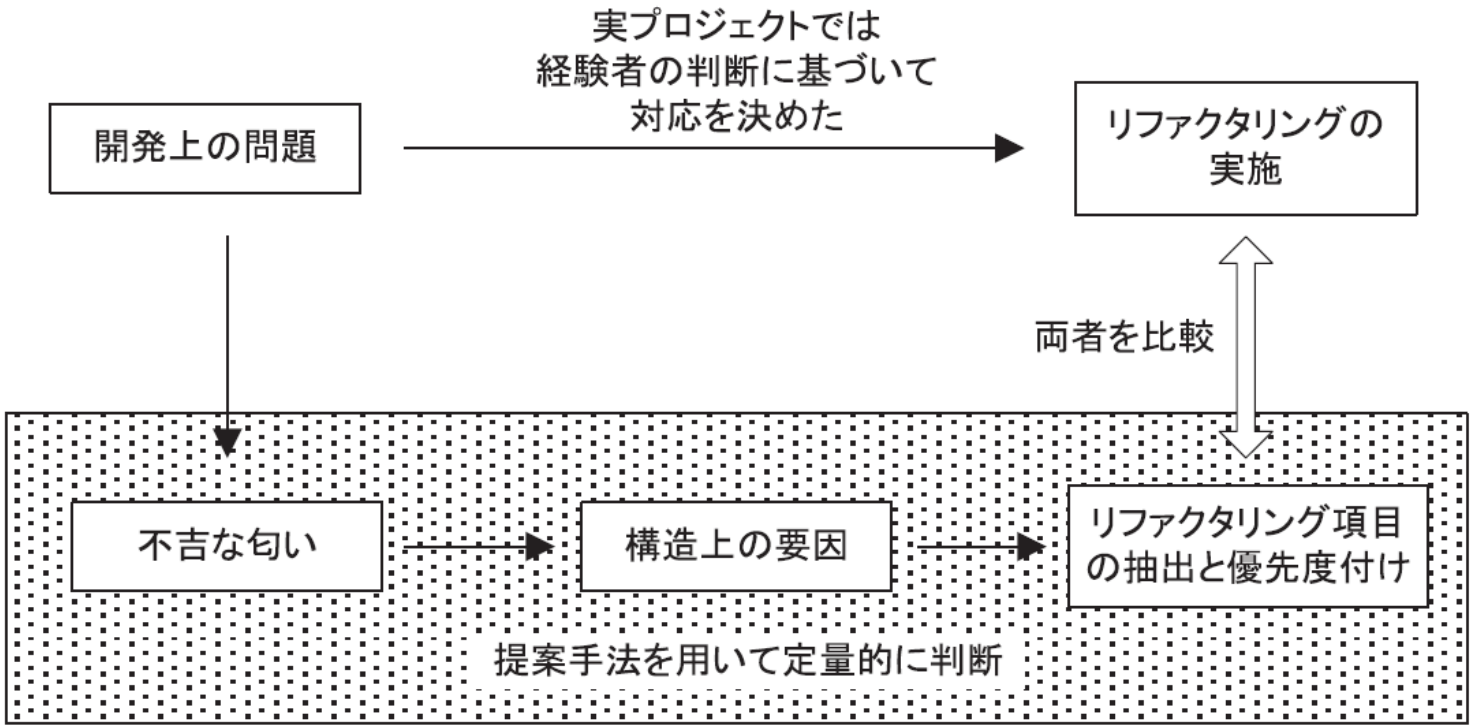
リファクタリング前



リファクタリング後

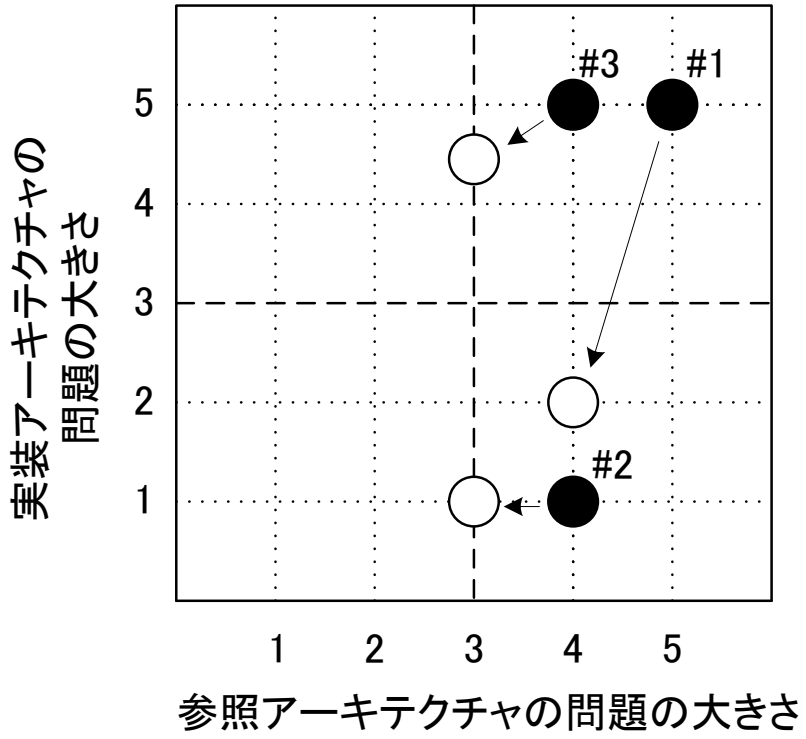
評価のアプローチ

- 専門家の判断によるプロジェクトでの実施内容と、提案手法を用いた分析結果を比較し、提案手法の妥当性を評価する。
  - ①専門家の判断と類似の判断が提案手法でできること
  - ②問題のポートフォリオが実際の傾向を反映していること



ポートフォリオ図（総合）

- リファクタリング前後のポートフォリオ分析



問題の大きさを総合すると減少傾向にある

II. 環境の変化と新たなメトリクス

アーキテクチャの位置づけの変化

- 環境の変化
  - 要求の多様化
  - 開発スタイルのアジャイル化

    ⇒後から出てくる要求にも対応する必要あり
- 支える技術
  - CI/CDの普及
  - 検証・テストの自動化
- 「進化的アーキテクチャ」の考え方
  - 変更しないもの　⇒　変更しやすいこと

クリスチャン・チセリ、ニール・フォード他著 島田浩二訳「進化的アーキテクチャ」2018、オライリージャパン

アーキテクチャに関する新しいメトリクス

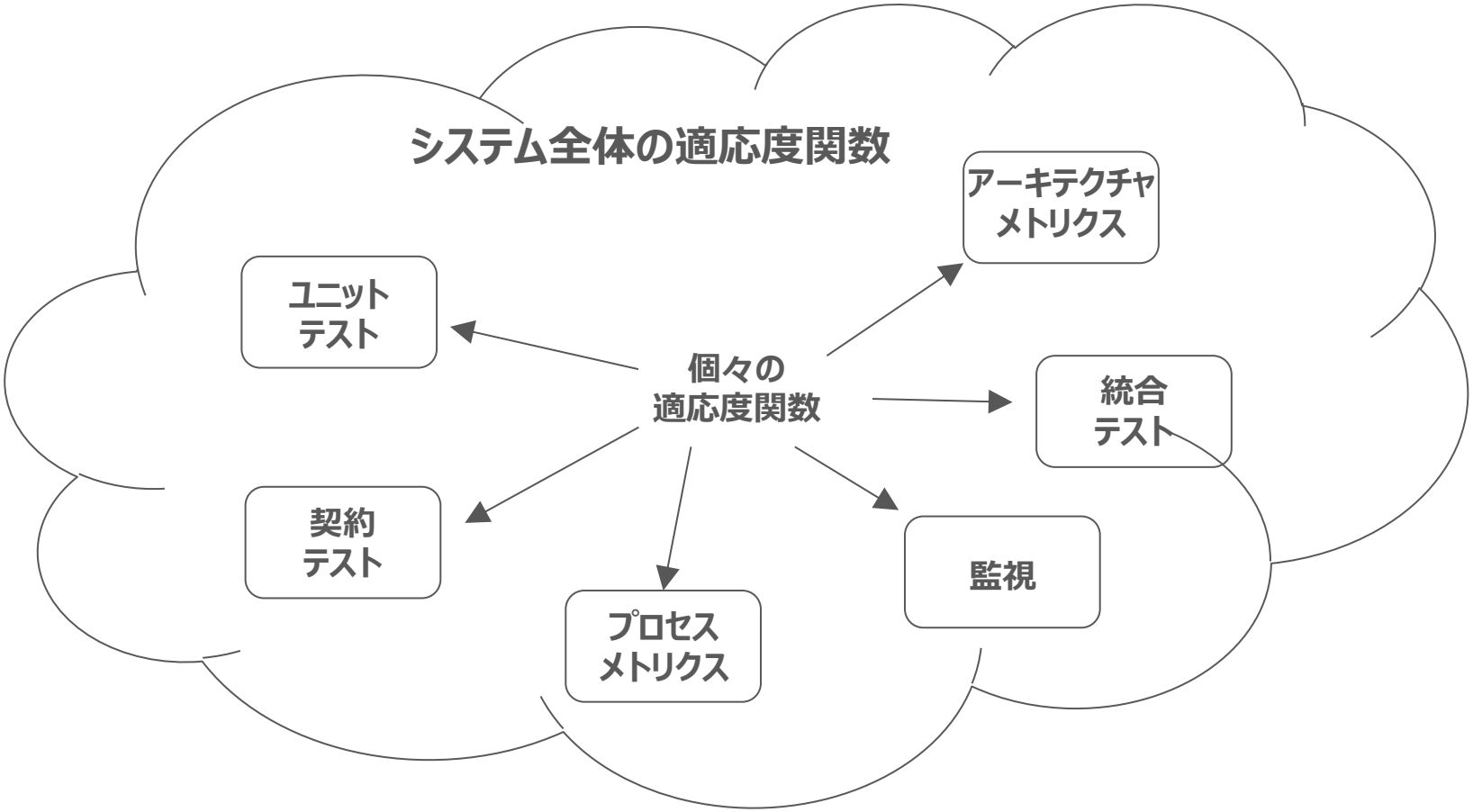
- 適応度関数
  - 実装の改善のために用いるKPIのようなもの
- モジュール性成熟度指数(MMI)
  - 参照アーキテクチャと実装アーキテクチャの総括的評価
- 構造負債指数
  - 循環依存関係を解消するために断ち切るべきリンクの数と依存関係の数を評価
- 影響度平均
  - ある任意の1要素に対して変更を加える場合に、影響を受ける要素の数の平均

⇒上記以外にも多数提案されている

ニール・フォード他著 島田浩二訳「ソフトウェア アーキテクチャ メトリクス」2024、オライリージャパン

適応度関数

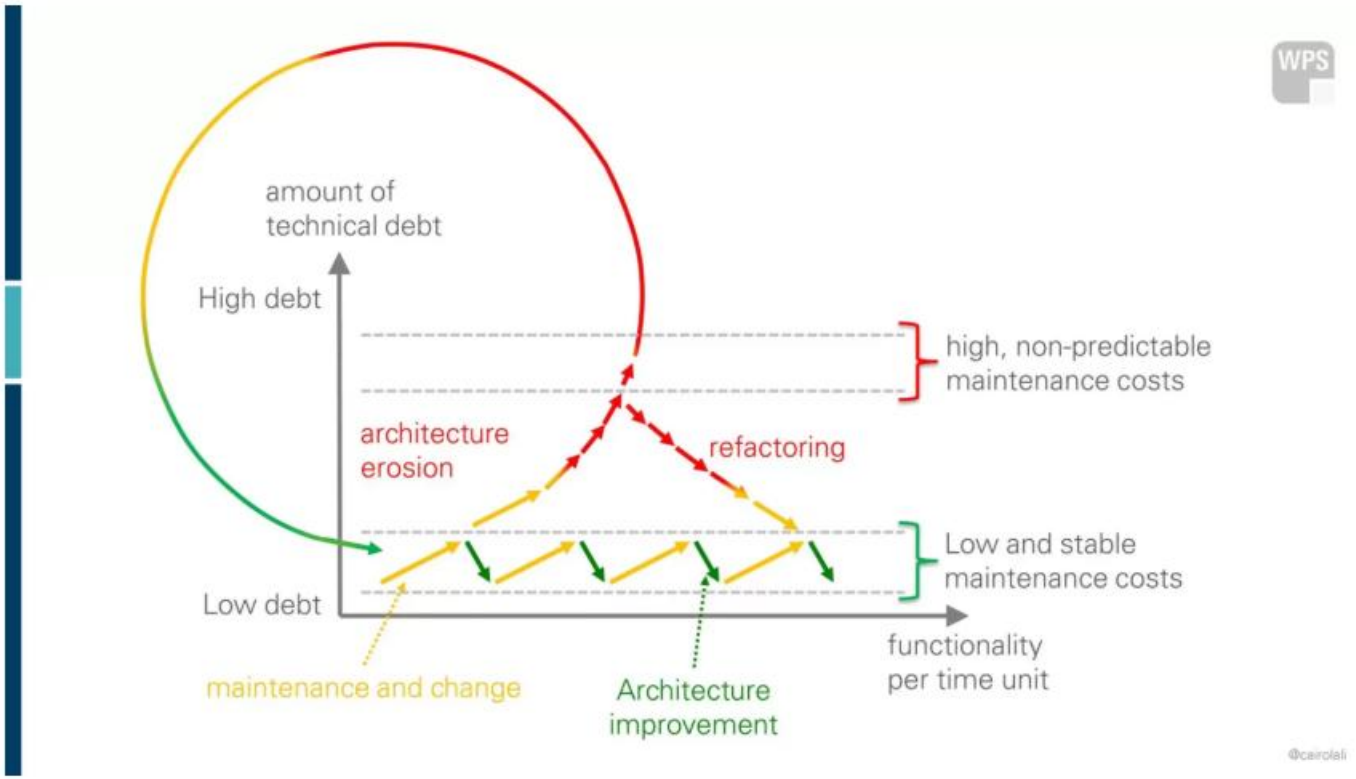
- 定義…見込みのある設計ソリューションが、設定した目的の達成にどれだけ近いかを要約するため使用する目的関数



クリスチャン・チセリ、ニール・フォード他著 島田浩二訳「進化的アーキテクチャ」2018、オライリージャパン、P21

モジュール性成熟度指数(1)

- あらかじめ定められたメトリクスを用いてアーキテクチャを評価し、リファクタリングの要否を判断する



https://apiumhub.com/tech-blog-barcelona/modularity-maturity-index/

モジュール性成熟度指数(2)

カテゴリ	サブカテゴリ	指標	～によって決定される
1.モジュール性(45%)			
	1.1. ドメインや技術のモジュール化(25%)		
		1.1.1. ドメインモジュールへのソースコードの割り当て(全ソースコードに占める割合)	アーキテクチャ分析ツール
		1.1.2. 技術レイヤーへのソースコードの割り当て(ソースコード全体に占める割合)	アーキテクチャ分析ツール
		1.1.3. ドメイン・モジュールのサイズ関係[(最大LOC/最小LOC)/数]	メトリクスツール
		1.1.4. 技術レイヤーのサイズ関係[(最大LOC/最小LOC)/数]	メトリクスツール
		1.1.5. ドメインモジュール、技術レイヤー、パッケージ、クラスは明確な責任を持つ	レビューア
		1.1.6. パッケージ・名前空間またはプロジェクトを介した技術レイヤーとドメインモジュールのマッピング	レビューア
	1.2. 内部インターフェース(10%)		
		1.2.1. ドメインモジュール・技術モジュールにインターフェイスがある(%違反)	アーキテクチャ分析ツール
		1.2.2. パッケージ・名前空間またはプロジェクトによる内部インターフェイスのマッピング	レビューア
	1.3. 比率(10%)		
		1.3.1. 大規模クラスにおけるソースコードの割合	メトリクスツール
		1.3.2. 大規模メソッドにおけるソースコードの割合	メトリクスツール
		1.3.3. 大規模パッケージのクラスの割合	メトリクスツール
		1.3.4. システムのメソッドのうち、高い循環複雑度を持つメソッドの割合	メトリクスツール

ニール・フォード他著 島田浩二訳「ソフトウエアアーキテクチャメトリクス」2024、オライリージャパン、PP.59-77

モジュール性成熟度指数(3)

2. 階層(30%)			
	2.1. ドメインや技術のレイヤー化(15%)		
		2.1.1. 技術のレイヤー化におけるアーキテクチャ違反の数(%)	アーキテクチャ分析ツール
		2.1.2. ドメインモジュールのレイヤ化におけるアーキテクチャ違反の数(%)	アーキテクチャ分析ツール
	2.2. クラス循環とパッケージ循環(15%)		
		2.2.1. 全クラスのうち、いずれかの循環に含まれるクラスの割合(%)	メトリクスツール
		2.2.2. 全パッケージのうち、いずれかの循環に含まれるパッケージの割合(%)	メトリクスツール
		2.2.3. 循環毎のクラス数	メトリクスツール
		2.2.4. 循環毎のパッケージ数	メトリクスツール
3. パターンの一貫性(25%)			
		3.1. パターンへのソースコードの割り当て(全ソースコードに対する割合)	アーキテクチャ分析ツール
		3.2. パターン間の循環依存関係(違反の割合)	アーキテクチャ分析ツール
		3.3. パターンの明示的なマッピング(クラス名や継承、アノテーションを介した)	レビューア
		3.4. ドメインやソースコードの分離(ドメイン駆動設計、ヘキサゴナルアーキテクチャ)	レビューア

ニール・フォード他著 島田浩二訳「ソフトウエアアーキテクチャメトリクス」2024、オライリージャパン、PP.59-77

モジュール性成熟度指数

実施結果

カテゴリ	サブカテゴリ	指標	～によって決定される	
1.モジュール性(45%)				
	1.1. ドメインや技術のモジュール化(25%)			
		1.1.1. ドメインモジュールへのソースコードの割り当て(全ソースコードに占める割合)	アーキテクチャ分析ツール	5
		1.1.2. 技術レイヤーへのソースコードの割り当て(ソースコード全体に占める割合)	アーキテクチャ分析ツール	5
		1.1.3. ドメイン・モジュールのサイズ関係[(最大LOC/最小LOC)/数]	メトリクスツール	8
		1.1.4. 技術レイヤーのサイズ関係[(最大LOC/最小LOC)/数]	メトリクスツール	8
		1.1.5. ドメインモジュール、技術レイヤー、パッケージ、クラスは明確な責任を持つ	レビューア	4
		1.1.6. パッケージ・名前空間またはプロジェクトを介した技術レイヤーとドメインモジュールのマッピング	レビューア	4
	1.2. 内部インターフェース(10%)			
		1.2.1. ドメインモジュール・技術モジュールにインターフェイスがある(%違反)	アーキテクチャ分析ツール	8
		1.2.2. パッケージ・名前空間またはプロジェクトによる内部インターフェイスのマッピング	レビューア	5
	1.3. 比率(10%)			
		1.3.1. 大規模クラスにおけるソースコードの割合	メトリクスツール	5
		1.3.2. 大規模メソッドにおけるソースコードの割合	メトリクスツール	5
		1.3.3. 大規模パッケージのクラスの割合	メトリクスツール	5
		1.3.4. システムのメソッドのうち、高い循環複雑度を持つメソッドの割合	メトリクスツール	6
2. 階層(30%)				
	2.1. ドメインや技術のレイヤー化(15%)			
		2.1.1. 技術のレイヤー化におけるアーキテクチャ違反の数(%)	アーキテクチャ分析ツール	4
		2.1.2. ドメインモジュールのレイヤ化におけるアーキテクチャ違反の数(%)	アーキテクチャ分析ツール	4
	2.2. クラス循環とパッケージ循環(15%)			
		2.2.1. 全クラスのうち、いずれかの循環に含まれるクラスの割合(%)	メトリクスツール	7
		2.2.2. 全パッケージのうち、いずれかの循環に含まれるパッケージの割合(%)	メトリクスツール	7
		2.2.3. 循環毎のクラス数	メトリクスツール	8
		2.2.4. 循環毎のパッケージ数	メトリクスツール	8
3. パターンの一貫性(25%)				
		3.1. パターンへのソースコードの割り当て(全ソースコードに対する割合)	アーキテクチャ分析ツール	4
		3.2. パターン間の循環依存関係(違反の割合)	アーキテクチャ分析ツール	4
		3.3. パターンの明示的なマッピング(クラス名や継承、アノテーションを介した)	レビューア	5
		3.4. ドメインやソースコードの分離(ドメイン駆動設計、ヘキサゴナルアーキテクチャ)	レビューア	5

⇒ MMI≒5.4

8～10:技術的負債の割合は低い
→ アーキテクチャが維持できている

4～８：既にかなりの技術的負債あり
→リファクタリングで対応できる可能性

４未満：多大な労力をかけなければ保守・拡張ができない → リプレースも検討すべき

III. 考察

適応度関数の観点

- 紹介事例では参照アーキテクチャと実装アーキテクチャの両方について適応度関数を用いて評価していた。
- 主には「依存関係」を適応度関数のパラメータとしてアーキテクチャのリファクタリングを推進していた。

モジュール性成熟度関数の観点

- プロジェクトでの「気づき」ではなく、あらかじめ評価項目がリストアップされているので、評価がしやすい
  - 多様なプロジェクトに一律の評価基準が当てはまるか？
- 参照と実装の問題が両方考慮されている
- 実プロジェクトではリファクタリングにより改善が実施できた
  - 評価結果は実態と合っていそう

今後の展開

- 生成AIの活用
- ツール化の検討
- リファクタリングorリプレースの判断基準

ご清聴ありがとうございました

テクマトリックス株式会社
ソフトウェアエンジニアリング事業部
牧　隆史

maki@techmatrix.co.jp
www.techmatrix.co.jp/product/quality.html