アーキテクチャ滑走路の ソフトウェアプロダクトライン開発への適用と評価

林 健吾[†1]　青山 幹雄[†2]

概要：近年のソフトウェアシステムの開発では，製品のグローバル化や市場要求の高度化，複雑化により，多様性と俊敏性の実現が競争の鍵となっている．本稿では SPLE（Software Product Line Engineering）の実践においてドメイン開発を俊敏化するために，アジャイル開発で提唱されているアーキテクチャ滑走路（AR：Architectural Runway）のコンセプトを自動車システムのソフトウェア開発に適用し，評価したケーススタディを報告する．AR を実装するに当たり，アプリケーション開発チームを AR 開発チームとして組織することで，人的資源と技術スキルの不足，市場投入の時間制約への要求に対応できるよう組織を設計した．本ケーススタディによって，SPLE を実践する組織がアーキテクチャを維持し，かつ，低コスト開発を継続できる SPLE 開発の実現へ貢献することを期待する．

An Architectural Runway for Software Product Line Engineering: Practical Application and Evaluation

KENGO HAYASHI[†1]　MIKIO AOYAMA[†2]

1. はじめに

SPLE（Software Product Line Engineering）は，低コスト，高品質でソフトウェアの製品系列（SPL：Software Product Line）における多様な製品を開発するアプローチである[23]．自動車ソフトウェア開発では Bosch のガソリンシステム開発において，市場セグメントごとに SPL を構築するなど，多くの実践例がある[26]．SPLE は，ドメイン開発とアプリケーション開発の 2 つの開発領域で SPL の製品の多様性に対応する[23]．ドメイン開発では SPL における共通性と可変性を分析してコア資産を構築する．共通性とは，SPL の製品間で不変である特徴点の集合である．可変性とは，SPL の製品間で差異が現れる特徴点の集合である．アプリケーション開発では，コア資産から個別製品を開発して市場に製品を供給する．

近年，自動車ソフトウェアを含むソフトウェアシステムの開発では俊敏な製品進化が求められるようになっている[10]．俊敏な製品進化に対しては，短いサイクルでインクリメンタルな開発を駆動する ASD（Agile Software Development）が実践されている[5][15][20]．Hanssen らは，SPLE と ASD はその実践において対立が生じることを報告している[11]．その一要因として，ASD が短期間のサイクルを短期間の軽量な計画で実行したい一方，SPLE では製品系列の多様性を要求分析する重量な計画フェーズを前もって設ける必要があることを挙げている．

製品の多様性とその俊敏な進化の両立に対応するために ASD と SPLE を統合した APLE（Agile Product Line Engineering）が実践されている[7][15]．APLE ではドメイン開発の軽量化が主要なテーマのひとつとなっている[7][24]．アプリケーション開発に先立って，多様性を分析するためにドメイン開発に長期間の投資を掛けることは，コスト超過や製品投入の遅延を招くリスクとなる．俊敏な製品進化への要求によって，ドメイン開発とアプリケーション開発を並行化する状況も生じており[14][28]，ドメイン開発のさらなる俊敏化が求められている．

APLE の実践におけるドメイン開発の俊敏化の要求は，SPL の運用においてアーキテクチャの崩壊[25]を招くリスクを高める．単一の SPL で要求の多様性を吸収できなくなると，複数の SPL の開発と運用が必要となる状況が生じ得る[1][26][27]．市場の要求に従って，既存の SPL から新たな SPL に成長させるとき，あるいは長期間 SPL の運用を継続すると，アーキテクチャ崩壊が起こり得る[25]．アーキテクチャ崩壊は，ソフトウェアの使用環境の変化において，ソフトウェアの開発組織が市場投入の時間制約の圧力を受け，アーキテクチャを維持するための技術的解決を適用するよりもフィーチャ（機能）開発の優先度が高くなることによって生じる[9]．アーキテクチャ崩壊は SPLE の運用の破綻を招くリスクとなることから，アーキテクチャ崩壊を予防し，維持するための方策が必要である．

Leffingwell は，大規模アジャイル開発のための SAFe（Scaled Agile Framework）フレームワークの中で，アーキテクチャを改善，拡張するためのコンセプトとしてアーキテクチャ滑走路（Architectural Runway，以下 AR）を提案している[19][20]．AR は，短期間のサイクルで反復してインクリメンタルに開発する ASD において，リファクタリングの頻度を抑え，現在および今後予期される要求を組み込

†1 (株)デンソー
DENSO CORPORATION
†2 南山大学
Nanzan University

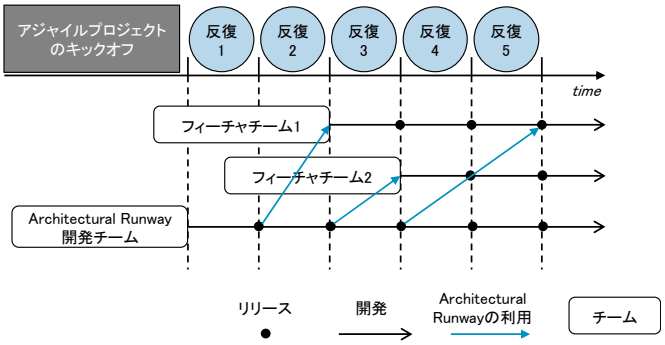

図 1 アーキテクチャ滑走路の概念図

Figure 1 Architectural Runway Overview

めるための既存の，または計画されたインフラであると定義されている．図 1 に，AR の開発と利用を時系列に表現した概念図を示す．顧客の要求を満たすことに集中したフィーチャチームとは独立に AR 開発チームを組織する．そして，フィーチャチームの開発を阻害しないように，インクリメンタル開発の中で継続的にアーキテクチャの維持と改善を推進する．

本稿では，SPLE のドメイン開発の俊敏化が求められる状況において，アーキテクチャの維持と改善を目的に AR のコンセプトを実装適用したケーススタディを報告する．AR を実装するに当たり，アプリケーション開発チームを AR 開発チームとして組織することで，人的資源と技術スキルの不足を補完し，市場投入の時間制約への要求に対応できるよう組織を設計した．本ケーススタディによって，APLE を実践する組織がアーキテクチャを維持した低コスト開発の継続可能な SPL 開発活動の実現へ貢献することを期待する．

本論文の構成は以下の通りである．2 章に SPLE のドメイン開発のアーキテクチャを維持するための関連研究を示す．3 章に本ケーススタディの開発のコンテキストを示す．4 章にて具体的な AR の実装方法と適用条件を示す．5 章と 6 章にて評価，考察を示す．7 章に本論文の結論をまとめる．

2. 関連研究

2.1 大規模アジャイル開発のアーキテクチャチーム

ASD では，開発者の人数が増大するに従って，成長していくソフトウェアを支えるために，専門チームを設けてアーキテクチャを改善する規律が採用されている．

大規模アジャイル開発のためのフレームワークである SAFe では，アーキテクチャの改造を推進する AR 開発チームを組織する[6][8][19][20]．AR 開発チームは，顧客の要求を満たすことに集中したフィーチャチームの開発を阻害しないように，インクリメンタル開発の中で継続的にアーキテクチャの維持と改善を推進する．

SAFe に対してより制約の少ない規律に基づく大規模アジャイル開発のフレームワークである LeSS でも，アーキテクチャの改造を推進する Undone 部門と呼ばれるチームが規定されている[6][8][16]．LeSS では 5〜9 人の開発チームが複数チームで連携して開発を推進する．Undone 部門は，それ以外の開発チームが周期的な開発サイクルであるスプリントの中でフィーチャ（機能）ごとの完了（Done）を目指すのに対して，非機能要求などの品質を確保するための横断的関心事に対応するために開発を支援する．

Spotify モデルは多チームでの製品開発の最適化を図る運用モデルである[3][8]．Spotify モデルは，スクワッド（部族），トライブ（分隊），チャプター（支部）に分割して組織を構成する．スクワッドは 9 人以下で構成された自己管理型チームである．トライブはスクワッドの集合であり関連した事業エリアを形成する．チャプターは特定の専門分野の集まりであり，スクワッドを横断したグループである．アーキテクチャなどの横断的関心事への対応方法を，チャプターを通して各スクワッドに行き渡らせる仕組みである．

各モデルにおいて，実装例としてのケーススタディの論文などは少ない[6]が，軽量なプロセスを用いて継続してアーキテクチャを維持する組織活動の枠組みが提供されている．本稿では，SAFe の AR のコンセプトを実装することで，SPLE におけるコア資産の可変性の維持・改善に取り組む．

2.2 APLE のドメイン開発の軽量化

Zhang らは，通信分野において，インクリメンタルでイテレーティブに SPL をリエンジニアリングするアプローチを実践している[28]．本ケーススタディでは，コンポーネントベースのアーキテクチャにおいて，初期製品として開発されたコンポーネント群を基に，ASD の規律に従って開発するチームが，各コンポーネントをリエンジニアリングしてコア資産としての可変性を構築する．ドメイン開発とアプリケーション開発を毎回の製品開発の ROI(Return On Investment)に基づく優先順位に従って小規模に分割することでドメイン開発の軽量化を果たしている．

Rumpe らは，コンポーネントベースのアーキテクチャにおいて，アプリケーション開発にて再利用コンポーネントを評価することで開発コストを最適化する方法を提案している[24]．コア資産として蓄えられたコンポーネントが，対象となる製品開発においてそのまま再利用できるか，可変性を追加する変更を加えるか，類似の別コンポーネントを開発するか，新規コンポーネントを開発するかを評価する．

Carbon らは，ASD による製品開発のスプリントにおいて，プランニングゲームにアプリケーション開発のアーキテクト，あるいはエンジニアが参加することで，再利用性を高める方法を提案している[4]．プランニングゲームは，スプリントでの開発アイテムを決定する軽量の計画フェーズである．インクリメンタルな製品開発の各スプリントでアプリケーション開発におけるフィードバックをイテレーティブに加えることで，可変性を備えたコア資産として蓄積可

能なソフトウェアを軽量に開発することを促す．

これらのケーススタディ，提案方法から，本稿では，コンポーネント単位でのアーキテクチャの改善，アプリケーション開発チームの参画を方策として取り組むことで，ドメイン開発における可変性構築の軽量化を実現する．

3. ケーススタディのコンテキスト

本章では，本稿のケーススタディのコンテキストを示す．コンテキストは，SPLE の評価フレームワークである BAPO モデルに沿って提示する[17][21]．BAPO は SPLE に対する以下 4 つの関心事の頭文字を並べた略語である．

(1) Business：SPL から利益を得る戦略，
(2) Architecture：ソフトウェアを構築する技術的な手段，
(3) Process：開発におけるロールや責務，関係，
(4) Organization：ロールや責務の組織構造への割当て．

3.1 Business ビュー

本稿のケーススタディは，自動車システムのサブシステムである，センサを利用した駆動制御システムの組み込みソフトウェア製品の開発が対象である．本製品はサブシステム内に選択的な複数機能をパッケージングしており，自動車のプラットフォーム上を流れる通信への適合と，車両の運動性能や車両骨格に基づく配置に適合するために，車両ごとに複数のバリエーションが生じる．

プロダクトの成長フェーズは 4 つの製品化フェーズを推移することが示されている[22]．新規製品として市場実績のない Innovation フェーズから始まり，市場が確立され製品発展の時間的速度が求められる Commercialization フェーズに至る．以降は，市場が成熟し新たな SPL は抑制され価格競争優位の Saturation フェーズを経て，製品がコモディティ化されて利益が縮小する Decline フェーズで終える．

本製品は Commercialization フェーズの早期に当たる．このフェーズの製品化戦略は，以降のフェーズに備えてドメイン開発を重視して新たな技術を組み込みながら，新たなアーキテクチャやコンポーネント開発を推進して製品の問題空間の安定を求めるものである[22]．本製品の戦略も，新たな機能を開発するために，SPL を成長させながら複数製品を生成する戦略をとる．

3.2 Architecture ビュー

SPLE では，生成される個々の製品アーキテクチャと，製品アーキテクチャを生成するためのプロダクトラインアーキテクチャの 2 階層でアーキテクチャが構成される．プロダクトラインアーキテクチャには次の 5 つのアーキテクチャのアプローチが存在する[22]．

(1) Architectural style：共通性や可変性は無視して単一の製品アーキテクチャによって SPL を実現する，
(2) Under-constrained architecture：Architectural style と異なり，バリエーションを生み出すためのアーキテクチャ上の強い制約を予め設計で確保しておく，
(3) Variance-free architecture：アーキテクチャは製品間の差異を考慮しない，詳細な記述としてのアーキテクチャ．製品の差異を設計と実装に委ねる，
(4) Parametric architecture：予め可変性を選択可能に設計しておき，インスタンスとしての製品をプロパティによって指定して引き出せるようにする，
(5) Service-oriented architecture for selective provisioning：提供するサービスをアーキテクチャの部分集合として予め規定しておき，各サービスをインスタンス化するプロセスを通して可変性を提供する．

本稿で対象とするソフトウェア製品では Parametric architecture を採用している．車両ごとの複数バリエーションをパラメータとして決定し，その組合せによって個々の製品を生成する．本アーキテクチャの利点は，個々の製品に対しての分析と計画の量が SPL の生成時点で見通しやすいことである．潜在的な問題は，パラメータの増設がプロダクトラインアーキテクチャにパラメータの組合せの複雑さとしての影響を与えやすいことである[22]．

プロダクトラインアーキテクチャを展開したソフトウェアの規模は 50k～100kLOC である．ソフトウェアの基本アーキテクチャはコンポーネントベースのアーキテクチャを採用している．可変点数は 100～200 で，年間約 150 のバリエーションの製品を生成している．

3.3 Process ビュー

3.3.1 SPLE としての開発プロセス

SPLE はドメイン開発とアプリケーション開発の 2 つのエンジニアリング領域が連携して多様性に対応する．これら 2 領域と SPL の運用方法には，次のようなプロセスの組合せが存在する[23]．

(1) 統合された開発プロセス：ドメイン開発とアプリケーション開発が統合されている．コア資産を生成するロールと個々の製品を生成するロールを分けない運用，
(2) 独立した開発プロセス：ドメイン開発とアプリケーション開発が独立している．コア資産をインタフェースとして，個々のロールを分けて独立した運用，
(3) 移譲した開発プロセス：ドメイン開発とアプリケーション開発が統合されている．SPL の確立と保守でロールを分け，コア資産を移譲した運用．

本ケーススタディはこれらの開発プロセスを複合した開発プロセスを採用している．SPL の確立までは統合された開発プロセスに従い，SPL 確立後は独立した開発プロセスに従う．コア資産はロール間で共有して運用している．

3.3.2 ASD としての開発プロセス

本ケーススタディは ASD における Scrum のフレームワークに基づく開発プロセスを採用している[5][12][13][14]．1～2 週間のスプリントで複数のチームが，チームごとに独立してイテレーティブに開発を行う．

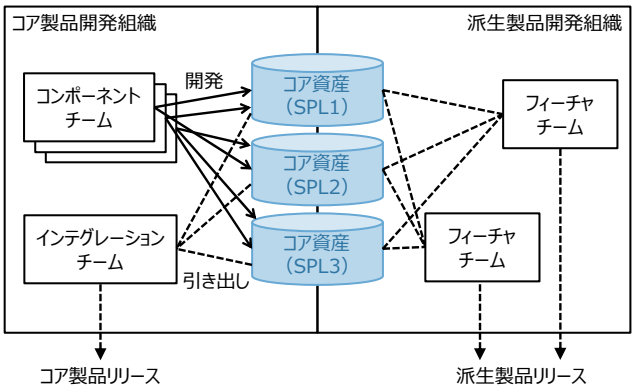

図 2 ケーススタディにおける組織への割当て

Figure 2 The Organization of the Case Study

3.4 Organization ビュー

3.4.1 SPLE としての組織の割当て

SPLE の Process に対して，本ケーススタディでは，図 2 に示すように，コア製品開発組織（以下，コア組織）と派生製品開発組織（以下，派生組織）の 2 つの並列した組織構造にロールと責務を割り当てている．コア組織は，SPL としてのコア資産の開発と，いくつかの初期製品をコア製品として開発し，統合された開発プロセスとしてドメイン開発とアプリケーション開発を併せて担う. SPL 確立後は，ドメイン開発の責務のみを担う．派生組織は，SPL 確立後にコア資産から個々の製品を派生製品として開発する責務を担う.

本ケーススタディでは複数のコア資産の中のひとつのコア資産の運用を対象とする．コア資産が複数設けられるのは，取り扱う製品が Commercialization フェーズの早期にあり，断続的な製品進化に対応するためである[1][10][22]. 断続的な製品進化と多様な製品展開に対応するためにも，コア組織と派生組織で分担する開発体制を採用している.

コア組織は 50～100 人規模の開発者で構成され，派生組織は 10～20 人規模の開発者で構成される. コア資産は SPL ごとに分割して管理運用し，3 つの SPL が存在している. 各 SPL は断続的な進化により開発の依存関係はなく独立して扱える. 本ケーススタディではひとつの SPL を対象としており，複数化している SPL のコンテキストを対象としない.

3.4.2 ASD としての組織の割当て

ASD の Process に対して，本ケーススタディではコア組織，派生組織のそれぞれでチーム編成を分けている．図 2 に本ケーススタディの組織への割当てを示す.

コア組織では 5～10 チームがコンポーネントチームとして編成されており，1 チームのインテグレーションチームがコア製品を生成する．コンポーネントチームは，アーキテクチャを構成するソフトウェアコンポーネントごとに独立して開発するチームである.

派生組織ではコンポーネントを横断して機能単位で開発を進める複数のフィーチャチームから成る. この構成は，一般的な ASD で推奨されているフィーチャチームの編成に従っている[19][20]. 派生組織では 2 チームがフィーチャチームとして編成されている．各チームはアプリケーション開発として，コア資産から生成する製品単位で開発を行う[14].

4. アーキテクチャ改善のための AR の実装

4.1 アーキテクチャ改善の動機

4.1.1 アーキテクチャ改善のタイミングと動機

本稿のケーススタディは，新たなハードウェアと追加機能に対応すべく，既存の SPL をベースとした新規の SPL を派生するための試作開発を対象としている．既存の SPL を派生して新規の SPL を開発する場合, 既存の SPL におけるアプリケーション開発で生じた技術的負債[9]に対してアーキテクチャ改善への動機が生じる．本ケーススタディでは，以下の 3 つの動機が生じた.

(1) アプリケーション開発を容易にする

派生組織でアプリケーション開発を複数対応する中で，構成管理などの開発ビュー[18]における改善点が発見される．例えば，複数製品の開発時期が重なった場合に，パラメータ設定用のファイルが製品ごとに独立して改造できるようにしたい，と言ったような動機である.

(2) 可変点の最適な設計を可能とする

既存の SPL に対する改造は，投資を最小限にして拡張性や保守性をトレードオフした設計が選択され易い．新規 SPL において，設計を最適化することで，アーキテクチャ崩壊の進行を緩めて運用コストを低下させたい.

(3) 可変点の補完を可能とする

Parametric architecture では，予め可変点がプロダクトラインアーキテクチャに織り込まれている方が，アプリケーション開発の見通しが立てやすい．新規 SPL が，既存の SPL の個々の製品開発と並行して開発されていても，既存の SPL 開発で生じた新たな可変点を取りこんでおきたい.

4.1.2 アーキテクチャ改善のレベル

アーキテクチャの改善には，アーキテクチャを構成する要素内，要素のいくつか，要素間の関係のどこまでを再構築するかに応じて Refactoring，Renovating，Rearchitecting の 3 つのレベルが存在する[2].

(1) Refactoring：アーキテクチャの断片的な成長により，コンポーネント間に循環的な依存が生じたり，過度な設計の一般化が生じたり，アーキテクチャ上の不備が評価された場合に，その後の拡張性を確保するために機能性を変えずに構造を改善する，

(2) Renovating：Refactoring による改善では効果的でないアーキテクチャ上の弱い要素に対して，Refactoring と補完的に，アーキテクチャの構成要素をひとつもしく

は複数置き換える形で再構築する，

(3) Rearchitecting : Refactoring や Renovating では対処できないレベルの，例えば技術プラットフォームの置き換えなどのビジネス上の重要な変化が生じた場合に，SWOT（Strength，Weakness，Opportunity，Threat）分析などを通じてアーキテクチャ全体を再構築する．

本ケーススタディでは，構成管理を含む開発ビューに対しては Renovating を，可変点の最適な設計と補完についてはアーキテクチャを構成する一部コンポーネントに対しての Refactoring のレベルのアーキテクチャ改善を対象とする．

4.2 アーキテクチャ滑走路の実現方法

4.2.1 アーキテクチャ滑走路実装の設計コンセプト

1 章と 2.1 で示した通り，AR は開発フレームワークにおけるコンセプトであり，実装方法は実践者，実践組織に委ねられている．本稿では，前節より SPLE における AR を，SPL の可変性を改善して SPL のサステイナビリティを向上させるインフラ，であるとして，本ケーススタディのコンテキストにおいてどのように実現可能であるかに取り組む．SPLE における AR を実現するために，アーキテクチャ改善の動機と改善レベル，組織構造から，アーキテクチャ滑走路の実現に対して以下 3 つの設計コンセプトを立案した．

(1) AR 開発チームは派生組織のチームを利用する

アーキテクチャ改善の動機に基づくと，アーキテクチャの改善は既存の SPL のアプリケーション開発のナレッジを所有していることが望ましい．アプリケーション開発のドメイン知識は派生組織のフィーチャチームに蓄えられている．フィーチャチームであれば，特定のソフトウェアコンポーネントに限定せず，アプリケーション開発で改造され得るソフトウェアコンポーネントを横断して，ドメイン知識を利用した開発が期待できる．

組織構造の面からも派生組織から AR 開発チームを組織する理由が生じる．コア組織は新規 SPL の開発に資源を集中させているため，AR 開発チームのために資源を割くことが難しいためである．

(2) アーキテクチャ改善のアーキテクトを擁立する

アーキテクチャの改善レベルを考慮して，アーキテクチャ改善のためのアーキテクトを定義して割り当てる．アーキテクチャの改善レベルとして，開発ビューに対する Renovating が存在している．開発ビューの更新は，ソフトウェアアーキテクチャ全体を取り扱うため，コア組織の全コンポーネントチームに統一的な変更を促すための設計と通知が必要となる．AR 開発チームが独立して活動するだけでは，コンポーネントチームの開発に混乱を招く恐れがある．また，開発ビューの設計にはアーキテクト固有のスキルが必要である．

(3) コンポーネントチームと改善の単位を同期する

アーキテクチャを改善するための AR 開発チームの活動は，コア組織のコンポーネントチームと改善の単位を同期する方針とする．コア組織はソフトウェアコンポーネントのサブセットでコンポーネントチームを組織している．また，アーキテクトが開発ビューの Renovating を定めた後は，Refactoring についてはコンポーネント単位で実行できることが見込まれる．AR 開発チームがソフトウェアコンポーネントを横断して活動した場合，複数のコンポーネントチームとコミュニケーションする必要が生じると，活動の効率が低下するリスクも生じる．改善の単位をコンポーネントチームと同期させることで，開発現場の混乱を最小化することが期待できる．

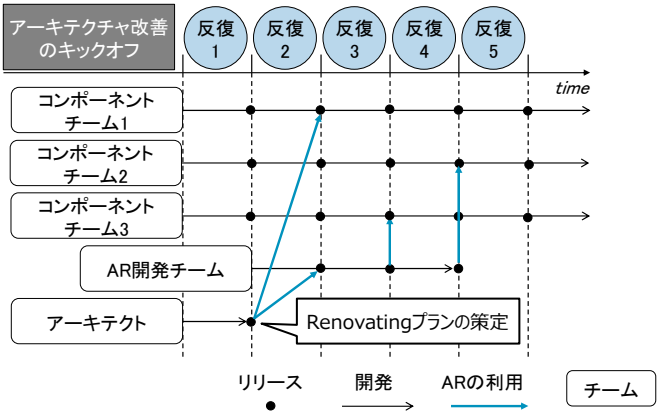

図 3 AR 開発プロセスの概要

Figure 3 Architectural Runway Development Process Overview

4.2.2 AR（アーキテクチャ滑走路）の実装

AR の設計コンセプトに従った，AR の実装を組織構成と開発プロセスと対応づけて図 3 に示す．

(1) 組織構成

本ケーススタディでは，AR 実装のための組織構成として，アーキテクト 1 名，AR 開発チームを 2 チーム組織した．アーキテクトは既存の SPL，新規の SPL のいずれのアーキテクチャも理解しており，開発ドメインに対して開発経験 4 年を有している．AR 開発チームは派生組織の 2 つのフィーチャチームをそのまま割り当てた．

コア組織のコンポーネントチームに対しては，新規の SPL を開発する体制のまま変更は加えていない．

(2) 開発プロセス

本ケーススタディにて実装した AR の開発プロセスの概要を図 3 に示す．AR の実装は，まずアーキテクトによる Renovating プランの策定から開始する．Renovating プランが策定できたら，プランに基づいて AR 開発チームおよび，個々のコンポーネントチームによるアーキテクチャの改善活動を開始する．ここで，コンポーネントチームは新規の SPL の開発を継続してイテレーティブに進めている．

Refactoring はコンポーネントチーム自身と AR 開発チームによって実行される．ソフトウェアコンポーネントによっては Refactoring の規模が十分に小さいため，コンポーネ

ントチーム自ら実行できるチームも存在した．Refactoring の規模が大きいソフトウェアコンポーネントについては，AR 開発チームが Refactoring を実行する．

AR 開発チームによる Refactoring は設計コンセプトに従い，コンポーネントチームと同期して以下の手順で実行する．

a) 対象のコンポーネントチームと AR 開発チームで Refactoring の同期をとるために，スプリントを開始するためのプランニングを共同で実行する．
b) スプリントの中盤で Refactoring の設計内容をコンポーネントチームにてレビューする．
c) スプリントの最後に Refactoring したソフトウェアコンポーネントのレビューを実施して改造コンポーネントをハンドオーバーする．

すべてのソフトウェアコンポーネントが，コンポーネントチームあるいは AR 開発チームによって Refactoring が完了した時点でアーキテクチャの改善を完了する．

4.3 ケーススタディにおける適用条件

本ケーススタディでアーキテクチャ滑走路を適用した条件を示す．

AR を適用した全体の期間は 20 週である．ASD としての反復単位であるスプリントは 2 週間で設定しており，全 10 スプリントで適用している．10 スプリント中 2 スプリントでアーキテクトによる Renovating プランを策定している．さらに 2 スプリントを掛けて，Renovating プランにおける構成管理のインフラを AR 開発チームにて構築した．残りの 6 スプリントにおいて，アーキテクチャ改善活動がコンポーネントチームと AR 開発チームによって実行した．

AR 開発チームによるコンポーネントの Refactoring は，4 つのソフトウェアコンポーネントを対象に，3 つのコンポーネントチームに対して実行された．AR 開発チームの活動は，既存 SPL に対するアプリケーション開発と並行するため，断続的に実行スプリントを計画して実行した．

5. 評価

5.1 評価方法

アーキテクチャ滑走路を適用した効果を以下の観点で評価した．

(1) アーキテクチャにおける可変性の改善，
(2) アーキテクチャ改善の達成度，
(3) アーキテクチャ改善の開発プロセスの並行性．

以下に，それぞれの観点での評価結果を示す．

5.2 アーキテクチャにおける可変性の改善

アーキテクチャにおける可変性の改善結果として，AR 適用前後における AR 開発チームによるソフトウェアコンポーネントと，ソフトウェア全体に対する可変点の変化を

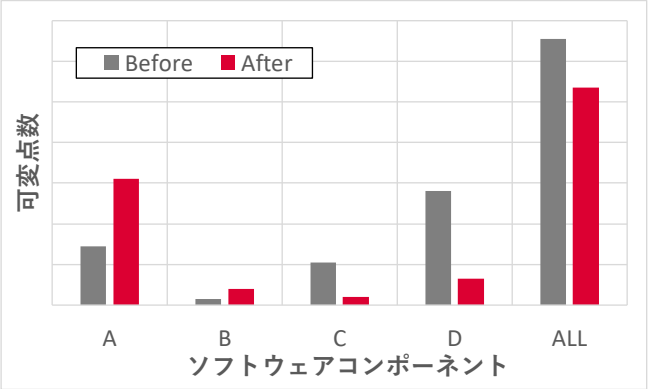

図 4 可変点数の変化

Figure 4 The Change of the Number of Variation Points

図 4 に示す．本ケーススタディでは，Refactoring と Renovating を対象としており，適用前後においてコンポーネントの構造は維持されている．そのため，適用前後で個々のコンポーネントの可変点数の変化を比較することで可変性の改善を評価する．

A，B のソフトウェアコンポーネントは適用前後で可変点数が増加している．A は可変点数が 2.1 倍，B は可変点数が 2.7 倍となっている．コンポーネント A はアーキテクチャとしてパラメータを統括しているコンポーネントであり，可変点の最適設計を図った結果として可変点数が増加した．コンポーネント B は，可変点の補完を図った結果として可変点数が増加した．

C，D のソフトウェアコンポーネントは，可変点の最適設計として，冗長であったパラメータ設定を削除した結果，可変点数がそれぞれ 81.0%，76.8%削減された．ソフトウェア全体としては 18.3%の可変点数が削減された．

AR の適用を通して，アーキテクチャにおける可変点数の削減と補完，集約により可変性が改善できたと評価できる．

5.3 アーキテクチャ改善の達成度

アーキテクチャ改善の達成度として，ソフトウェアコンポーネントにおいて Refactoring が必要なコンポーネントの内，Refactoring が達成されたコンポーネントの割合を評価した．結果として，100%のコンポーネントを Refactoring することができた．

Renovating プランにおいて，構成管理のための改造を含めた Refactoring に必要なソフトウェアコンポーネント数は 10 コンポーネントであった．その内，AR 開発チームにて Refactoring されたコンポーネントは 4 コンポーネントで，残り 6 コンポーネントはコンポーネントチームにより Refactoring され，すべてのコンポーネントがプラン通りに Refactoring された（図 5）．

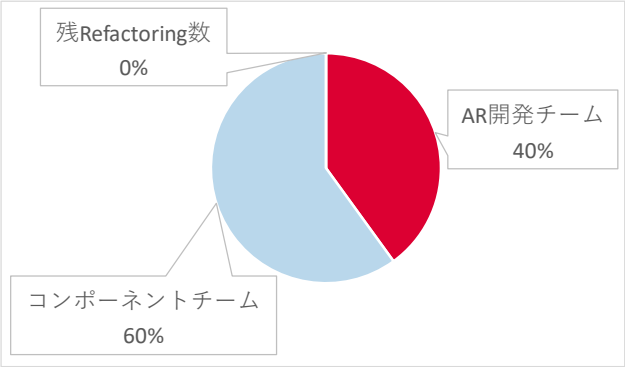

図 5 Refactoring されたコンポーネントの内訳
Figure 5 Breakdown of Refactored Software Components

このことから，改善後のソフトウェアアーキテクチャはモレなく設計を反映できたことで一貫性を備え，サステイナビリティ（sustainability：維持性）の高いアーキテクチャ改善ができたと言える[2].

5.4 アーキテクチャ改善の開発プロセスの並行性

アーキテクチャ改善の開発プロセスの並行性について定性的に評価する. AR を適用した 10 スプリントにおいて，アーキテクト並びに AR 開発チームは，コンポーネントチームと独立性高く開発を進めることができた.

2 週間の各スプリントにおいて, AR 開発チームからコンポーネントチームに求めた工数は，平均 1 時間のプランニングと，平均 2 時間の設計レビュー，平均 2 時間のハンドオーバー（レビューを含む）の時間工数であった．各チームの人数は考慮せず，1 日 8 時間の工数とした場合，1 スプリント 10 日間で 80 時間費やせるとして，コンポーネントチームは 6.3%の時間を AR 開発チームに投資した.

AR を適用してアーキテクチャを改善するために，コア組織が投資する工数は 10%以下である結果が得られた.

6. 考察

6.1 AR（アーキテクチャ滑走路）の有効性

6.1.1 アーキテクチャ改善への有効性

5.2，5.3 で示した通り，本ケーススタディでは AR の実装はアーキテクチャ改善に対して有効に機能した．アーキテクチャ改善への有効性としては，アーキテクトの任命による要因と，AR コンセプトによる要因に分けられる.

アーキテクトを選任で設ければ，アーキテクチャを改善するための方策を立案することは可能となる．5.2 で示したような可変性の改善の効果の程度はアーキテクトのスキルに依存して現れるものである．しかし，アーキテクトを任命するだけでは，立案した方策を遂行完了することが難しい．立案した方策はソフトウェアの様々な要素に適用する必要があり，相応の工数が求められるためである.

アーキテクトが立案するアーキテクチャの改善には構成管理方法の変更を伴う，開発ビューの変更も含まれていた．これは対象コンポーネントの可変点の実装方法を変更した上で，ソフトウェアの構築方法を変更する必要があった．5.3 で示したように，対象となるすべてのコンポーネントが Refactoring されることが，その後の開発としてのサステイナビリティを高めることとなる．達成度の向上は，AR コンセプトを適用した効果と言える．事項で本効果について示す.

6.1.2 人的資源とスキルの有効活用

AR コンセプトを利用することで，組織の人的資源とスキルを有効活用することに成功し，5.3 で示したようなアーキテクチャ改善の達成度を向上できた．組織の人的資源とスキルが有効活用できた理由を考察する.

本ケーススタディにおいて，AR 開発チームによって担った 4 つのソフトウェアコンポーネントは，アプリケーション開発において改造する機会の多いコンポーネントであった．これらのコンポーネントはその他のコンポーネントに対して，アーキテクチャ改善の量の大きいコンポーネントでもある.

アプリケーション開発のドメイン知識を有した AR 開発チームが対応することで，小さいコストで Refactoring を実行することができている．AR 開発チームが担当したコンポーネントは，アーキテクチャ改善の方策に基づいて，Refactoring のためのコンポーネントの設計分析を必要としていた．コア組織で設計分析するコストを確保することは難しい．また，この分析は，アプリケーション開発において頻繁に改造を繰り返す部品に集中していた．Refactoring 後はアプリケーション開発で開発し易い実装であることが望ましい．アプリケーション開発の経験者が参画することで効率的に期待値に適した実装を得ることができる.

単純に方策を展開すればよいコンポーネントはコンポーネントチームにて，分析を要するコンポーネントは AR 開発チームにて分担することで，効率的に対象のすべてのコンポーネントを Refactoring することができ，サステイナビリティの高い改善を果たすことができたと考えられる.

6.1.3 市場要求への時間制約への対応性

5.4 で示した通り，AR のコンセプトを適用することで，SPLE のドメイン開発の機能開発を推進する資源の消費を抑えて，アーキテクチャ改善を実行することができた．市場要求への時間制約に対しては，ドメイン開発の担当組織がフィーチャの実現に集中できることで有利に働く. また，初期製品投入後の多様性への対応はアーキテクチャ改善によって開発コストが小さくなり，短期間での市場投入に対しても有利となる.

AR を実装することで，市場要求への時間制約への対応性は高くなる効果が期待できる.

6.2 AR（アーキテクチャ滑走路）の適用条件

6.2.1 アーキテクチャスタイル

本ケーススタディでは，Parametric architecture に基づい

たソフトウェアにおいて，AR の有効性が果たされることが確認できた．その他のアーキテクチャスタイルに対して有効であるかを考察する．

Architectural style の場合，プロダクトラインアーキテクチャと製品アーキテクチャは同一であり，単一組織によって実現され得る．この場合，AR 開発チームは同一組織内で確保するか，新たな投資で生み出す必要が生じ，人的資源の有効活用の効果は低下する可能性が高い．

Under-constrained architecture，Variance-free architecture の場合，Parametric architecture と同様にアプリケーション開発が個別に生じやすく，ドメイン知識の蓄積が期待される．本ケーススタディと同様の効果が期待できる．

Service-oriented architecture for selective provisioning の場合，アプリケーション開発はインスタンス化するプロセスを実行するためのドメイン知識が蓄えられやすい．また，サービス単位でドメイン知識が蓄えられる傾向にあるため，Architectural style と同様に人的資源の有効活用の効果は低下する可能性がある．

アプリケーション開発でソフトウェア全体のドメイン知識を得らえるか否かが，本ケーススタディの実装形式によって AR による有効性を得るひとつの条件だと言える．

6.2.2 組織と開発プロセス

6.2.1 で考察した通り，組織としてアプリケーション開発を繰り返してドメイン知識を蓄える組織構造と開発プロセスを採用している場合に，本ケーススタディの形式での AR 実装の効果は高くなると考えられる．組織分割されていることで，ドメイン知識を有した組織を利用して人的資源とスキルを活用することが可能となる．

従来の ASD と同様にドメイン開発がフィーチャチームの集合で構成されている場合は，ドメイン開発担当組織単独で AR 開発チームを規定することが可能となるが，アプリケーション開発のスキルは活用し難いことが予想される．SPLE として可変性の改善を期待する場合は，アプリケーション開発のドメイン知識を活用する方策を取り入れることが望ましい．

本ケーススタディでは，アプリケーション開発の資源を活用している．アプリケーション開発の量が増大し，アプルケーション開発チームの資源の流動性を確保できない場合には，本方法による AR の実装は困難となる．

6.2.3 スケーラビリティ

本ケーススタディは 5〜10 チームレベルの開発において適用している．ASD としての LeSS において，LeSS Huge に移行する基準上の開発規模であり大規模には到達していない[6]．大規模の開発において，AR のコンセプトの適用が可能であるか考察する．

AR によるアーキテクチャ改善が可変性の向上であり，対象がソフトウェアコンポーネントの Refactoring に分解できる場合は，大規模の開発でも対応可能であることが考えられる．コンポーネントに活動が閉じていれば，AR 開発チームの数を増やすことで対応することが可能となるためである．

AR によるアーキテクチャ改善したい可変性がコンポーネントを横断した依存性を備える場合は対応が複雑化することが予測される．ソフトウェア全体に対して複雑な依存性を備えている場合，可変性の分析に基づく Renovating として網羅性を確保することが難しくなるためである．この場合，サステイナビリティの確保の効果が限定されるため，投資対効果を考慮した継続的な活動の運用方法を立案する必要があると考えられる．

6.3 AR（アーキテクチャ滑走路）の効果の分析

6.3.1 AR コンセプト非適用時の仮説

本ケーススタディでは適用時のみを評価対象としており，提供できる考察が限定されている．AR コンセプトを適用しなかった場合に本コンテキストの開発がどのような状態となるかの仮説を立てることで評価を補完する．

AR のコンセプトを適用しない場合，コンテキストとして生じ得るのは，アーキテクチャ改善を SPL 確立までにコア組織単独で実行するか，SPL 確立後に派生組織単独で実行するか，という状況である．コア組織単独でじっこうする場合，改善コストが小さなコンポーネントは改善を実行し得るが，改善コストが大きなコンポーネントは実行が難しくなり，サステイナビリティの向上は限定的となる．派生組織単独で実行する場合，組織資源も不足する可能性が高く，またチーム数が限られることで改善に掛かる期間も長くなる．その結果，改善は見送られて個々のアプリケーション開発のコストは相対的に大きくなり，サステイナビリティは低下することが想定される．

AR コンセプトを非適用とした場合は，SPLE のサステイナビリティはコア組織のスキルに単純依存して，初期のアーキテクチャと実装の結果を反映するか，コア組織の資源量に応じて小改善するに留まる仮説が立てられる．

6.3.2 AR コンセプトと実装方法の効果の分担

AR は，リファクタリングの頻度を抑え，現在および今後予期される要求を組み込めるための既存の，または計画されたインフラである，という定義がされたコンセプトである．AR コンセプトによる効果は，ソフトウェアの開発を継続する中でアーキテクチャを改善することで，その後の開発を可能にする，もしくは開発コストを低減するところに現れる．本ケーススタディにおける効果としては，5.2 において可変性が向上したという結果，5.3 において 40% の AR 開発チームによるアーキテクチャ改善の達成，5.4 の並行性が実現できている状態に現れていると考えられる．

SPLE を前提とした，本ケーススタディにおける実装方法による効果は，主に 6.1.2 で考察した人的資源とスキルの有効活用に現れていると考えられる．SPLE ではドメイン開発とアプリケーション開発という 2 つの開発領域を備

えることで，資源とスキルの分化が生じ易い．AR 開発チームをアプリケーション開発の組織から編成することは，従来の ASD を前提とした AR コンセプトの実装では想定されていない前提条件となる．本状況を利用した実装方法が，資源とスキルの有効活用の効果を生み，アーキテクチャ改善のやり切りに繋がることで，SPL のサステイナビリティの向上に貢献したと考えられる．

7. まとめ

SPLE のドメイン開発の俊敏化が求められる状況において，アーキテクチャの維持と改善を目的に AR（アーキテクチャ滑走路）のコンセプトを適用したケーススタディを報告した．AR を実装するに当たり，アプリケーション開発チームを AR 開発チームとして組織することで，人的資源と技術スキルの不足を補完して，市場投入の時間制約への要求に対応できるよう組織とプロセスを設計し実装する方法を示せた．本ケーススタディによって，AR のコンセプトを利用することで，アーキテクチャとしての可変性やサステイナビリティを改善することの成功例を示すことができた．

参考文献

[1] M. Aoyama, Continuous and Discontinuous Software Evolution, Proc. of IWPSE 2001, ACM, Sep. 2001, pp. 87-90.
[2] P. Avgeriou, Architecture Sustainability, IEEE Software, Vol. 30, No. 6, Nov./Dec. 2013, pp. 40-44.
[3] D. Barton, et al., One Bank's Agile Team Experiment, Harvard Business Review, Vol. 96, No. 2, Mar./Apr. 2018, pp. 59-61.
[4] R. Carbon, et al., Providing Feedback from Application to Family Engineering, Proc. of SPLC 2008, IEEE, Sep. 2008, pp. 180-189.
[5] M. Cohn, Agile Estimating and Planning, Prentice Hall, 2005.
[6] L. Craig, et al., Practices for Scaling Learn & Agile Development, Addison-Wesley, 2010.
[7] J. Díaz, et al., Agile Product Line Engineering- A Systematic Literature Review, Software: Practice and Experience, Vol. 41, No. 8, Jul. 2011, pp. 921-941.
[8] T. Dingsøyr, et al., Agile Software Development, Springer, 2010.
[9] Z. Durdik, et al., Sustainability Guidelines for Long-Living Software Systems, Proc. of ICSM 2012, IEEE, Sep. 2012, pp. 517-526.
[10] C. Ebert and J. Favaro, Automotive Software, IEEE Software, Vol. 34, No. 3, May/Jun. 2017, pp. 33-39.
[11] G. K. Hanssen, et al., Process Fusion, J. of Systems and Software, Vol. 81, No. 6, Jun. 2008, pp. 843-854.
[12] K. Hayashi, et al., Agile Tames Product Line Variability, Proc. of SPLC 2017, ACM, Sep. 2017, pp. 180-189.
[13] 林 健吾, 青山 幹雄, マルチプロダクトライン開発における可変性の構造分析に基づくアジャイルアプリケーション開発方法の提案と評価, SES 2017 論文集, 情報処理学会, Aug.-Sep. 2017, pp. 190-197.
[14] 林健吾, 青山 幹雄, アジャイルプロダクトライン開発におけるポートフォリオ駆動開発モデルと管理方法の提案と適用評価, SES 2018 論文集, 情報処理学会, Sep. 2018, pp. 157-165.
[15] P. Hohl, et al., Searching for Common Ground, Proc. of ICSSP 2017, ACM, Jul. 2017, pp. 70-79.
[16] M. Kalenda, et al., Scaling Agile in Large Organizations, J. of Software: Evolution and Process, Vol. 30, No. 10, Oct. 2018, pp. 1-25.
[17] K. Kang, et al., Feature-Oriented Product Line Engineering, IEEE Software, Vol. 19, No. 4, Jul./Aug. 2002, pp. 58-65.
[18] P. Kruchten, The 4+1 View Model of Software Architecture, IEEE Software, Vol. 12, No. 6, Nov. 1995, pp. 42-50.
[19] D. Leffingwell, Agile Software Requirements, Addison-Wesley, 2011.
[20] D. Leffingwell, SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering, Addison-Wesley, 2016.
[21] F. van der Linden, et al., Software Product Family Evaluation, Proc. of SPLC 2004, LNCS Vol. 3154, Springer, Aug.-Sep. 2004, pp. 110-129.
[22] M. Mattsson, Software Product Line Architecture, J. Bosch (ed.), Software Architecture - An Overview of the State-of-the-Art, Research Report, University of Karlskrona/Ronneby, 1998, pp. 63-70.
[23] K. Pohl, et al., Software Product Line Engineering, Springer, 2005.
[24] B. Rumpe, et al., Agile Synchronization between a Software Product Line and its Products, Informatik 2015, LNI Vol. 246, Springer, Sep.-Oct. 2015, pp. 1687-1698.
[25] L. De Silva, et al., Controlling Software Architecture Erosion, J. of Systems and Software, Vol. 85, No. 1, 2012, pp. 132-151.
[26] M. Steger, et al, Introducing PLA at Bosch Gasoline Systems: Experiences and Practices, Proc. of SPLC 2004, LNCS Vol. 3154, Springer, Aug.-Sep. 2004, pp. 34-50.
[27] B. Tekinerdogan, et al., Supporting Incremental Product Development using Multiple Product Line Architecture, Int'l J. of Knowledge and Systems Science, Vol. 5, No. 4, Oct.-Dec. 2014, pp. 1-16.
[28] G. Zhang, et al., Incremental and Iterative Reengineering towards Software Product Line, Proc. of ICSM 2011, IEEE, Sep. 2011, pp. 418-427.