エンタープライズアジャイル並行開発に適したソフトウェアアーキテクチャ評価方法の提案と評価

田中 優之[1]　青山 幹雄[2]

概要：複数機能の並行開発を可能とするエンタープライズアジャイル（EA）の実践事例が数多く報告されている．実践時の問題点として，開発中の機能間の依存関係を原因とする同一ソースファイル更新（コンフリクト）が発生し開発遅延が生じることが挙げられる．これに対処するための方法としてプロダクトと開発組織を統括するためのリリースマネジメント技術と適切なソフトウェアアーキテクチャ設計が考えられる．本稿ではソフトウェアアーキテクチャ評価方法 SAAM for Enterprise Agile（SAAM for EA）を提案し，スマートフォン向けアプリケーションに適用する．評価結果から，EA 開発に適したソフトウェアアーキテクチャ評価方法に必要な特性の一つとして複数機能のインクリメンタル開発能力が挙げられた．EA を導入する組織は SAAM for EA を用いることでスケジュール遅延の回避が可能になる．本稿は EA 開発組織と SA の関係を考察し EA 導入時にサービス指向アーキテクチャ（Service-Oriented Architectures: SOA）を適用することを提案する．

キーワード：エンタープライズアジャイル，並行開発，ソフトウェアアーキテクチャ，アーキテクチャ評価

A Software Architecture for Concurrent Development of Enterprise Agile Method and Its Evaluation

MASAYUKI TANAKA[†1]　MIKIO AOYAMA[†2]

1. はじめに

複数機能の並行開発を可能とするエンタープライズアジャイル開発(Enterprise Agile 開発，以下 EA 開発と略記)の実践事例が数多く報告されている．それぞれの実践事例は導入した EA 開発フレームワークがその組織において機能したことを示す．一方で，他の組織において同様にその EA 開発が機能するかを判断することは難しい．EA 開発を実践する組織への調査結果から，導入時の課題として組織の文化, EA 開発フレームワークの複雑さ，ソフトウェアアーキテクチャ(以下 SA と略記)の不適合などが挙げられている[14].

本稿では，EA 開発と SA の関係に着目する．全体が一つのモジュールで設計されているモノリシックアーキテクチャは EA 開発に適していないことが報告されている[14]. EA 開発は並行開発を前提としているので，実践時に同一ソースファイルの更新（コンフリクト）が発生しやすいことが一つの原因である[16]．この課題に対処する方法としてプロダクトと開発組織をリードするマネジメント技術(リリースマネジメント)と適切な SA 設計がある．しかし Scaled Agile Framework (SAFe)[8][11]，Large Scale Scrum (LeSS), LeSS Huge[10]などの実践事例が報告されている EA 開発向けフレームワークではリリースマネジメントについての議論はあるが SA に関する議論は少ない．本研究では EA の並行開発時に発生するコンフリクト問題に着目し SA がインクリメンタル開発に適したものであるかどうかを評価する方法を提案する．この評価方法を用いることでスケジュール遅延の回避や SA 変更による効果シミュレーションが可能になる．

2. 研究課題

本研究の課題をまとめる．以下の 2 点となる．

(1) EA 開発に適した SA 評価方法の持つべき特性は何か
(2) EA 開発のマネジメントを支援する開発プロセスの可視化は可能か

3. 関連研究

3.1 EA 開発フレームワーク

アジャイル開発は小さな一つのチームで実施することから時間を経て複数のチームで大規模に開発する EA 開発へ進化をしてきた．Scaled Agile Framework（SAFe），Large Scale Scrum（LeSS），Scrum of Scrums など様々な EA 開発フレームワークが提案されておりその事例が毎年報告されている[3]．EA 開発フレームワークの開発プロセスは一つのチームでアジャイル開発をする時と同様に，短い期間に規模が小さなリリースを繰り返すことで変化に対応する．EA 開発では複数のチームが並行して開発を行う必要があり，各 EA 開発フレームワークではタスクの優先度付けを

1 南山大学 大学院 理工学研究科 ソフトウェア工学専攻
Graduate Program of Software Engineering, Nanzan University
2 南山大学 理工学部 ソフトウェア工学科
Dep. of Software Engineering, Nanzan University

行う仕組みを提供している．これはプロダクトオーナが担う役割の一つであり，LeSS，LeSS Huge ではこの優先度付けをするツールとしてプロダクトバックログを利用する．各 EA 開発フレームワークは優先度付けを行うための打ち合わせを実施することを定義している．図 1 に EA のプロダクトオーナの役割を示す[15]．図 1 はプロダクトオーナの役割が要求の把握（図中 Conflicting Business Needs），ステークホルダとのコミュニケーション（図中 Sponsor Intermediary），優先度付け（図中 Groom Prioritizer），そしてリリース管理（図中 Release Master）など多岐に渡ることを示す．SAFe はより大規模な EA 開発を実施するため組織の資産管理方法（ポートフォリオマネジメント）を提案している．各 EA 開発フレームワークは大規模なアジャイル開発を行うため多くのルールを定義するがルールの複雑さは導入時の課題となっている[14]．

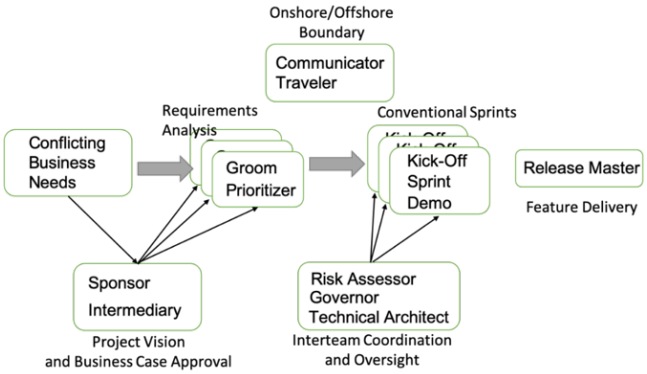

図 1 EA のプロダクトオーナの役割

EA 開発ではリリースマネジメントも課題の一つである．短い期間に複数のチームで並行して開発とリリースを繰り返すので，それらを支援する技術と環境が必要である．具体的な技術としてビルド自動化，TDD(Test Driven Development)[1]，マイクロサービスアーキテクチャ[13]などが挙げられる．これらを組み合わせることで短い期間にリリースを繰り返すことが可能となる．一方で EA 開発フレームワークは開発プロセスのフレームワークであるのでこれらについて述べられることはない．これは EA 開発フレームワークの導入が困難である一要因と考えられる．

3.2 並行開発を支援する技術

(1) ドメインの概念を導入した SA

MVC や MVP，レイヤードアーキテクチャの SA はプレゼンテーション層，ビジネスロジック層などを機能分割のためのコンポーネントとして設計するドメイン独立な SA である．この境界に応じたチーム編成を行い，複数機能を並行開発することを考える．このときドメイン独立な SA はチームを横断して作業調整をする必要がある．一方，ドメインの概念を導入した SA はチームを横断した作業調整は必要ない．それはドメイン内の実装に限定されるという理由による[7]．ドメインの概念を導入した SA として Clean Architecture（図 2）[12]がある．Clean Architecture はドメインを定義するモデリング方法 Domain-Driven Development（DDD）[6]を実現する方法の一つである．Clean Architecture を iOS アプリケーション開発時に利用する方法として VIPER（図 3）が提案されている．

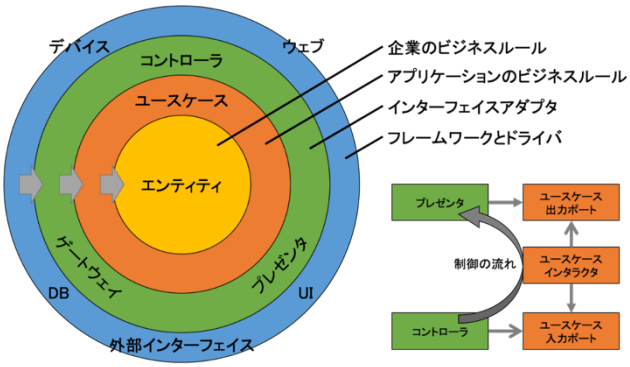

図 2 Clean Architecture

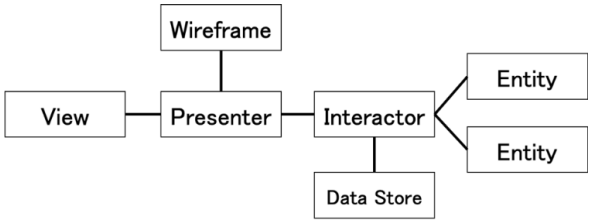

図 3 VIPER

(2) サービス指向アーキテクチャ（Service-Oriented Architectures: SOA）

SOA はアプリケーションフロントエンド，サービス，サービスリポジトリ，サービスバスから構成される SA である[9]．マイクロサービスアーキテクチャは SOA を実現する方法の一つであり，ドメイン単位でサービスを分割する．ドメイン間は疎結合となり，HTTP 通信などを用いた連携を行うので利用する技術の選択が柔軟になる[7]．その結果，サービスとチーム構造を一致させることが可能となりドメインごとにリリースが可能となる[13]．マイクロサービスアーキテクチャは並行開発を支援する技術である．

3.3 Scenario-Based Architecture Analysis Method（SAAM）

SAAM は用意したシナリオの集合に対してスコアリングを行うことで評価対象の SA の変更容易性や拡張性を評価する方法である[1]．SAAM は簡潔な評価方法であり学習コストが低く，実施しやすいという特徴を持つ．一方で評価結果がシナリオに依存する点が課題である．

4. アプローチ

本稿では，EA 開発に適した SA を特定するための SA 評価方法を提案し，スマートフォンアプリケーションに適用し評価方法の有効性，妥当性を示す．評価方法の観点として SA 観点に加えてマネジメント観点を導入し，両方の観点から評価するアプローチを提案する．これは，次の 2 つの仮説が本稿で提起した EA 並行開発時のコンフリクト問

題の解決の鍵であることによる．

(1) SA 観点の仮説：きわどいコンポーネントの粒度を必要十分なサイズにすることが一般に並行開発でのコンフリクトの頻度を最小化する．

(2) マネジメント観点：きわどいコンポーネントの改修を行う時の並行開発の実行を管理することがコンフリクトの頻度を最小化する．

5. SAAM for EA (Scenario-Based Architecture Analysis Method for Enterprise Agile)

5.1 評価方法の概要

EA において課題となる並行開発に適した SA の評価を行うため SAAM for EA を提案する（図 4）．SAAM for EA は SAAM と同様にシナリオに基づき SA を評価する．

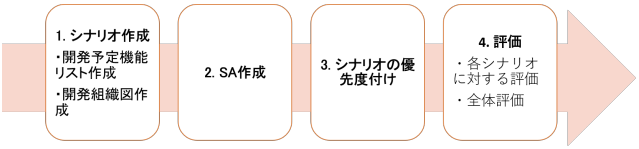

図 4 SAAM for EA 概要

5.2 評価シナリオの設計

評価シナリオは開発予定の機能リストを組み合わせ，開発組織を定義することで作成する（図 5）．作成された評価シナリオは各開発チームが開発を担当する機能を示す．これは複数の機能を並行開発するシナリオとなる．SAAM の評価結果は評価に使用するシナリオの質に影響を受ける課題がある．SAAM for EA は開発予定の機能リストと開発組織を事前に定義することで SAAM の課題であるシナリオ作成を補助する．

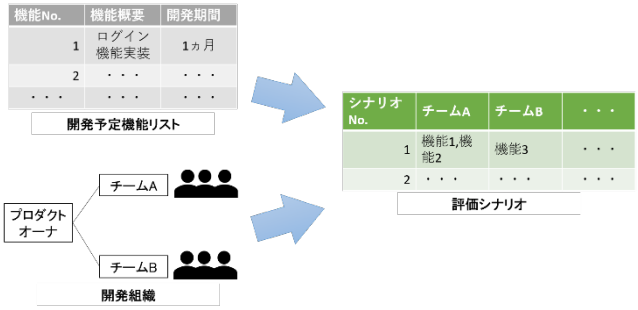

図 5 評価シナリオの設計

5.3 並行開発の可視化によるシミュレーション

SAAM for EA では開発の並行実行の視覚化のために並行開発シミュレーション図（図 6）を提案する．並行開発シミュレーション図はリリースマネジメント作業を可視化することで EA 並行開発時のコンフリクト問題のシミュレーションを可能とする．図中の横軸は時系列を示し，縦軸は開発予定の機能を示す．並行開発シミュレーション図はコンフリクトに着目することで開発中の機能間の依存関係を時間軸と共に可視化する．図 6 ではスプリント 1 において機能 (1) は他の機能開発と依存することなくリリースできることを示す．機能 (2)，(3)，(4)はスプリント 1 において依存関係がありコンフリクトを解消した後にリリースが可能であることを示す．機能 (5) は開発期間が 2 スプリント必要であり，スプリント 1 で開発をしていた機能または機能 (6) に依存していることを示す．機能 (6) は開発期間が 2 スプリント必要であり，スプリント 1 で開発をしていた機能または機能(5)に依存していることを示す．機能(5)，(6)はコンフリクトを解消した後にリリースをすることが可能であることがわかる．並行開発シミュレーション図を用い機能の開発順序を検討することでスケジュール遅延を最小限にする開発順序をシミュレーションすることができる．様々な開発順序を検討した結果，依存関係を最小化することが難しい場合はインクリメンタル開発に適さない SA であることがわかる．

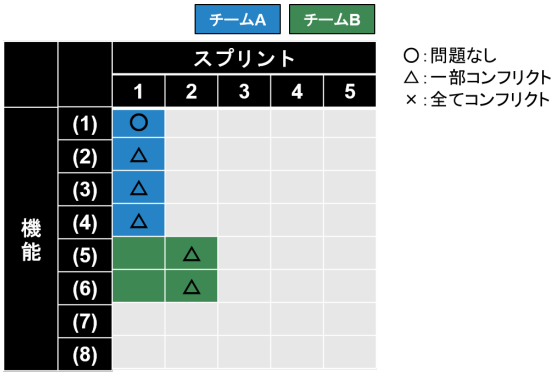

図 6 並行開発シミュレーション図

5.4 評価

評価作業は SAAM と同様に各シナリオに対する評価を実施することに加え，並行開発シミュレーション図を用いた評価を各シナリオに対して実施する．各スプリントでコンフリクトの発生の有無を確認し，機能間の依存関係を時間軸と共に評価する．コンフリクトが多く発生する場合はそのシナリオにおいてインクリメンタル開発が困難であることを示す．

6. 評価対象ソフトウェア

6.1 機能

評価対象は京都の観光情報を提供するスマートフォンアプリケーションとする[17][18]．スマートフォンアプリケーションが持つ主な機能を表 1 に示す．

表 1 評価対象ソフトウェア機能リスト

No.	機　能
1	地図が表示され，地図面の操作が可能
2	任意の地点への徒歩ナビゲーション
3	京都市からのお知らせ情報ページ
4	観光地の情報を地図上で確認できる
5	近くの飲食店を検索することができる

6.2 SA

MVC を用いた SA を図 7 に示す. Model を担当するコンポーネントを桃色, Controller を担当するコンポーネントを橙色，View を担当するコンポーネントを緑色で示した．VIPER を用いた SA を図 8，図 9 に示す．VIPER を用いた SA は図 8 に示すようにコンテキストが近いコンポーネントをまとめた．図 8，図 9 では MVC における Model 相当のコンポーネントを桃色, Controller 相当のコンポーネントを橙色，View 相当のコンポーネントを緑色で示した．

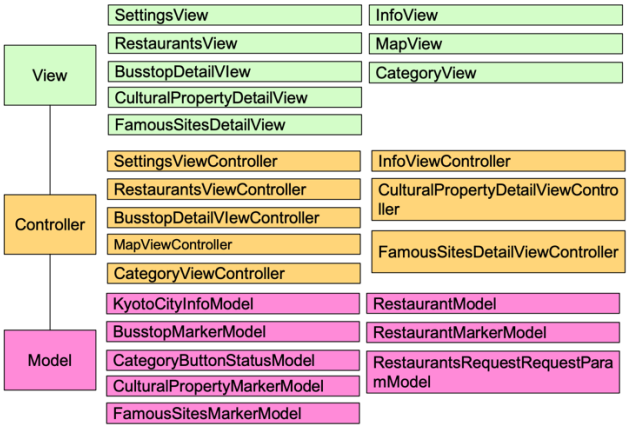

図 7 評価対象ソフトウェア（MVC）

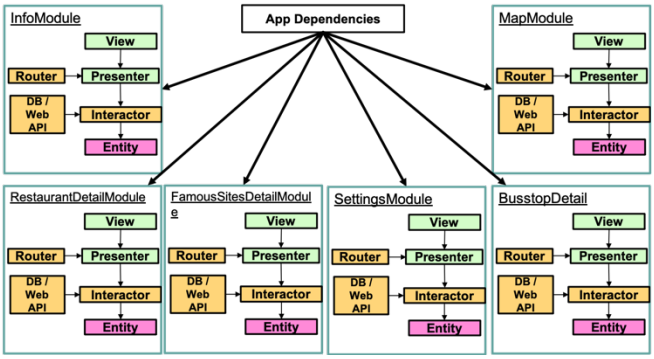

図 8 評価対象ソフトウェア（VIPER）

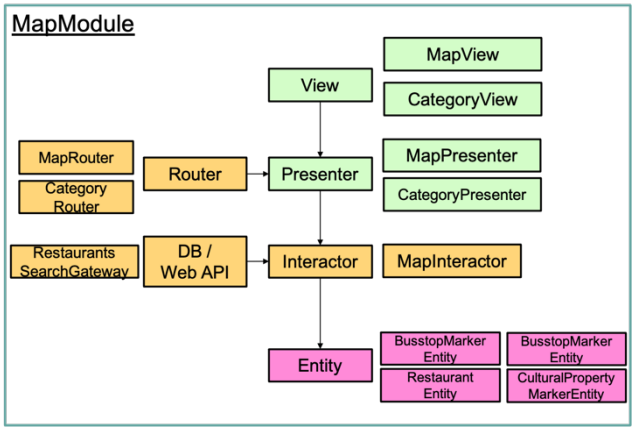

図 9 地図に関連するコンポーネント群（VIPER）

地図機能に関係するコンポーネントに着目することで MVC と VIPER の SA の違いを確認する．MVC における Controller のコンポーネントは MapViewController のみである（図 7）．VIPER における Controller のコンポーネントは MapInteractor，MapRouter など複数存在する（図 9）．このことから VIPER は MVC と比べて各コンポーネントの責務が小さくなることがわかる．

7. 評価

SAAM for EA を用いた評価を評価対象ソフトウェアに対して実施する．図 10 に示す開発組織（LeSS Huge）において表 2 に示す機能リストを組み合わせて作成した評価シナリオ（表 3）を用い評価を行う．エリアは大規模なアジャイル開発を行うために LeSS Huge が定義する概念である．開発チームはエリアに所属し，担当する機能の開発を進める．5.2 で述べたように開発組織（図 10）と開発予定の機能リスト（表 2）を組み合わせることで複数の機能を並行開発する評価シナリオとなる．なお，表 2 に示した開発予定期間は 1 スプリントを 2 週間として算出した．

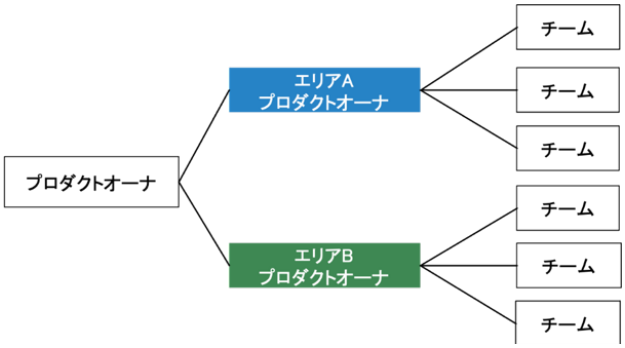

図 10 評価シナリオ（開発組織）

表 2 評価シナリオ（開発予定の機能）

機能番号	機能	開発予定期間(スプリント)
(1)	イベントの情報をプッシュ通知で配信する	1
(2)	お気に入りの観光地をローカルストレージに保存可能	1
(3)	ユーザ登録およびログインができる	1
(4)	地図デザインの変更	1
(5)	WebAPI（飲食店情報検索 API）の切り替えおよび仕様変更への対応	2
(6)	観光地の検索機能	2
(7)	アプリ起動時にお知らせダイアログを表示する	1
(8)	地図 SDK の変更	3

表 3 評価シナリオ

シナリオ No.	エリア A	エリア B
1	(1)，(2)，(3)，(4)を開発	(5)，(6)を開発
2	(1)，(2)，(3)，(4)，(7)を開発	(5)，(6)を開発
3	(1)，(3)，(5)を開発	(2)，(4)，(6)を開発
4	(1)，(2)，(3)，(7)を開発	(4)，(8)を開発
5	(1)，(3)，(5)を開発	(4)，(8)を開発

評価前の各シナリオの並行開発シミュレーション図を図 11 から図 15 に示す. MVC と VIPER それぞれの場合において評価を実施する.

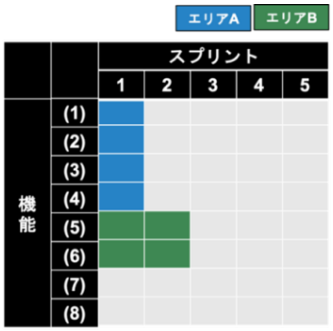

図 11 並行開発シミュレーション図（シナリオ 1）

図 12 並行開発シミュレーション図（シナリオ 2）

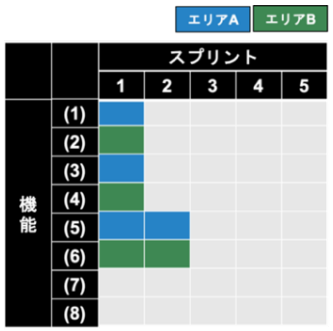

図 13 並行開発シミュレーション図（シナリオ 3）

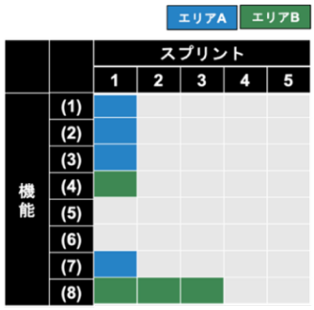

図 14 並行開発シミュレーション図（シナリオ 4）

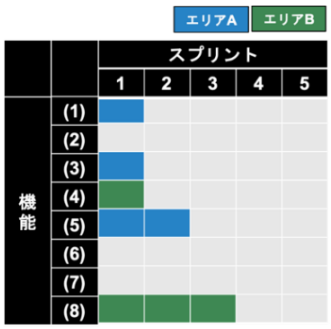

図 15 並行開発シミュレーション図（シナリオ 5）

8. 評価結果

8.1 SA 評価結果

表 4 に各シナリオの評価結果を示す. 表中，評価欄△は一部クラスファイルのコンフリクト発生を示す. 表 4 から全てのシナリオでコンフリクトが発生していることがわかる.

表 4 機能リリース時のコンフリクト比較

シナリオ No.	MVC		VIPER	
	評価	コンフリクトするクラス	評価	コンフリクトするクラス
1	△	MapViewController MapView	△	MapPresenter MapInteractor
2	△	MapViewController MapView	△	MapView MapPresenter MapInteractor
3	△	MapViewController	△	MapPresenter MapInteractor
4	△	MapViewController	△	MapView MapPresenter
5	△	MapViewController	△	MapPresenter

図 16 にシナリオ 4 における各コンポーネントの変更範囲を示す. MVC では MapViewController が変更対象のクラスになっており，VIPER では MapView と MapPresenter が変更対象となっている. これは VIPER では各コンポーネントの責務が小さくなっていることが原因である. 責務の小さいコンポーネントで発生しているコンフリクトは解消が容易である. 同様のことがシナリオ 2 とシナリオ 3 においても確認できた.

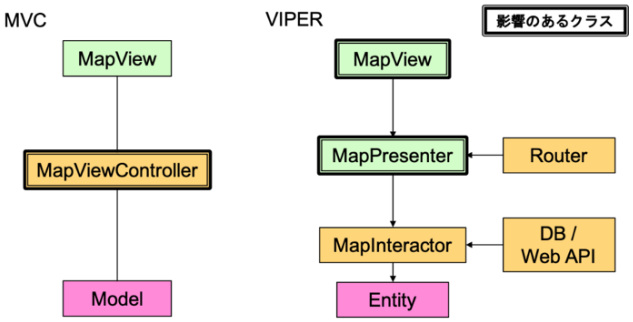

図 16 シナリオ 4 における比較

表 5 SA 評価結果

シナリオ No.	MVC	VIPER
1	0	0
2	-	+
3	-	+
4	-	+
5	0	0

これらを踏まえ，評価結果を表 5 に示す．表中の+は相対的に変更が容易であることを示し，-は相対的に変更が困難であることを示す．0 は差がないことを示す．VIPER がシナリオ 2，3，4 において MVC よりも良い結果となった．なお，シナリオ 1 とシナリオ 5 については差を確認することはできなかった．

8.2 並行開発シミュレーション図による評価結果

図 17 にシナリオ 4 における MVC と VIPER の並行開発シミュレーション図を示す．表中評価欄は〇をコンフリクトなし，△を一部クラスファイルのコンフリクト発生，×を全てのクラスファイルのコンフリクト発生を示す．シナリオ 4 のスプリント 1 において MVC は機能(4)のリリースが VIPER よりも容易であった．

図 18 は地図機能に関係するコンポーネントを示す．図中左側は MVC，右側は VIPER の SA である．機能(2)に関する変更箇所に着目する．MVC では変更箇所は MapViewController のみであり，VIPER では変更箇所は MapView，MapPresenter，MapInteractor など複数のコンポーネントに存在することがわかる．その結果，機能(2)の変更箇所は機能(4)の変更箇所と重なることとなった．VIPER は各コンポーネントの責務が小さく設計されているので複数のコンポーネントへ修正が発生する．これはコンフリクトが発生する原因となる．他のシナリオにおいても MVC の方がコンフリクトを少なくリリースできるケースを確認した．その理由はシナリオ 4 におけるケースと同様である．

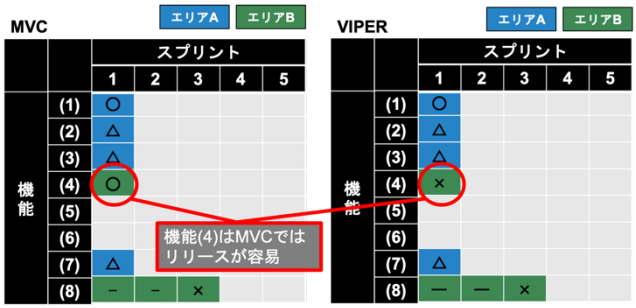

図 17 シナリオ 4 における並行開発シミュレーション図

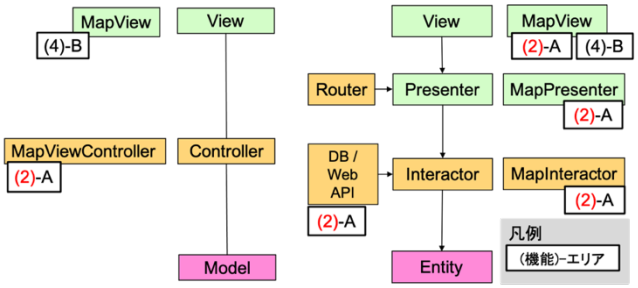

図 18 シナリオ 4 スプリント 1 における変更箇所の比較

9. 考察

9.1 研究課題（1）に対する考察

EA において並行開発を実施する際の課題から着想し SAAM for EA による評価を行なった．並行開発シミュレーション図はコンフリクトに着目することで開発中の機能間の依存関係を時間軸と共に可視化するので，複数機能のインクリメンタル開発に適した SA かを評価可能である．本稿は MVC と VIPER を比較する評価を実施した．MVC は VIPER と比較してコンポーネントの粒度が大きいため VIPER よりモノリシックな SA であると捉えることができる．モノリシックな SA は EA 開発に適さないことが EA 開発を実践する組織への調査から報告されている[14]. 表 5 に示す評価結果はモノリシックな SA（MVC）はコンポーネントの粒度が小さい SA（VIPER）と比較して EA 開発に適さないことを示した．このことから EA に適した SA 評価方法に必要な特性の一つは，複数機能のインクリメンタル開発を実施可能か評価できることと考えられる．SAAM for EA は評価シナリオ作成方法の提案と並行開発シミュレーション図を新たに導入することでそれを可能とした．

9.2 研究課題（2）に対する考察

図 1 で示したようにプロダクトマネージャの役割は優先度付けとリリース管理である．プロダクトにとって最適なリリーススケジュールを導き出すために，プロダクトマネージャは複数のシナリオを比較する．このとき，プロダクトマネージャは開発を行っている機能間の依存関係を考慮し最適なシナリオを選択する．

本稿提案の並行開発シミュレーション図はコンフリクトに着目することで開発を行っている機能間の依存関係を

可視化する．これは EA 開発におけるプロダクトマネージャがリリーススケジュールを検討する時に必要とする情報と同様である．このことから，並行開発シミュレーション図を用い開発中の機能間の依存関係を可視化することで EA 開発においてプロダクトマネージャが担うマネジメント作業を支援可能と考える．

9.3 EA 開発フレームワークの選択と SA

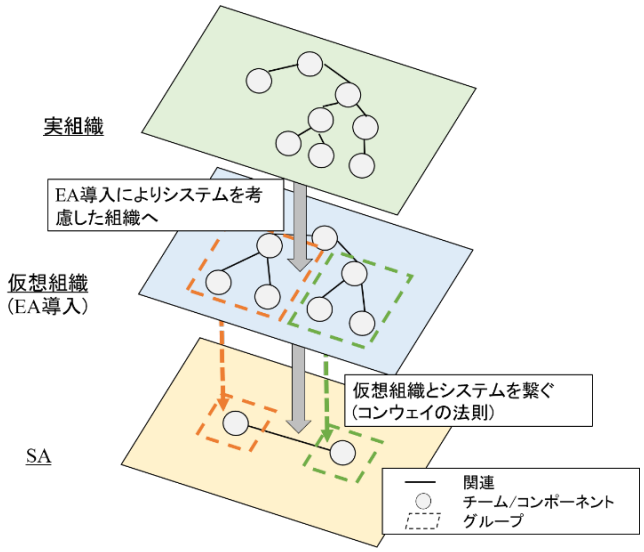

図 19 EA と SA の関係

同じシナリオにおいて MVC と VIPER で結果が異なるケースを確認した．これは SA の構造と開発組織の構造の違いによる結果と考えられる．図 19 に開発組織と SA の構造の繋がりを示す．設計を行う組織の構造がシステムの構造に反映されること[7]を考慮すると，EA 開発フレームワークの導入はシステムの構造に合わせた仮想組織の構築手段と考えられる．SAFe5.0 は実際の組織を第 1 の組織，EA 組織を第 2 の組織（仮想組織）と捉えている[19]．EA 開発フレームワーク選択時に重要なことは，組織が担当する SA の構造に合う EA 開発フレームワーク導入によって仮想組織を構築することである．

9.4 組織構造と SOA

9.3 で述べたように組織と SA の構造を合わせることが EA の選択時に重要である．このとき基準として SOA で提案される SA の利用が考えられる．SOA を導入している場合，その構成要素に対応する形で EA を導入することで組織構造と SA の関係を合わせることができる（図 20）．SOA を導入できておらず SA を分割できていない場合(モノリシックアーキテクチャ)はそれに合わせ一つのアジャイル開発チームとすることがリリースマネジメントの課題を最小限に抑えると考えられる（図 21）．また，マイクロサービスアーキテクチャを導入している場合は，より大規模な EA を導入できる可能性がある（図 22）．アプリケーションフロントエンドはドメインの概念を導入した Clean Architecture などを導入することで，同様のアプローチが可能と考えられる（図 23）．

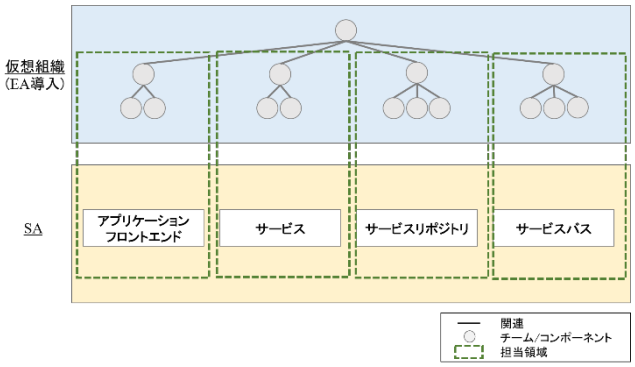

図 20 組織と SA（SOA）

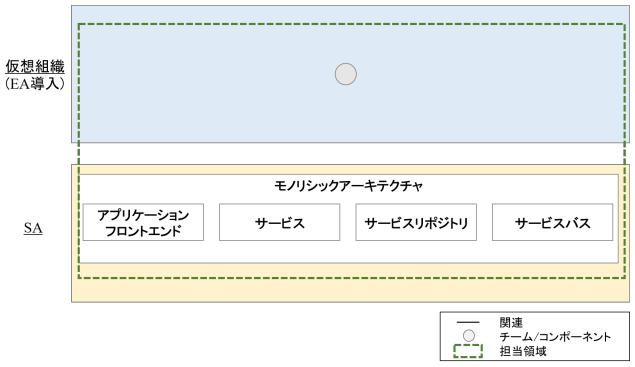

図 21 組織と SA（モノリシックアーキテクチャ）

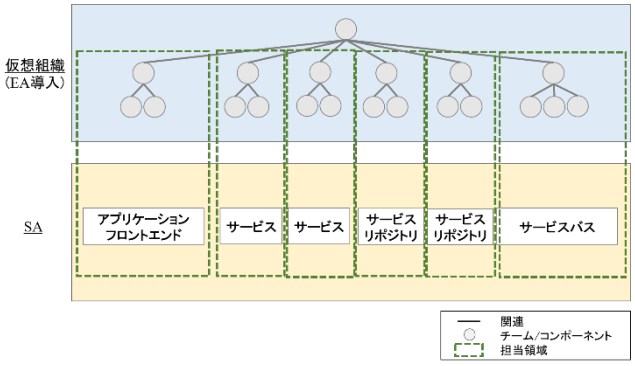

図 22 組織と SA（マイクロサービスアーキテクチャ）

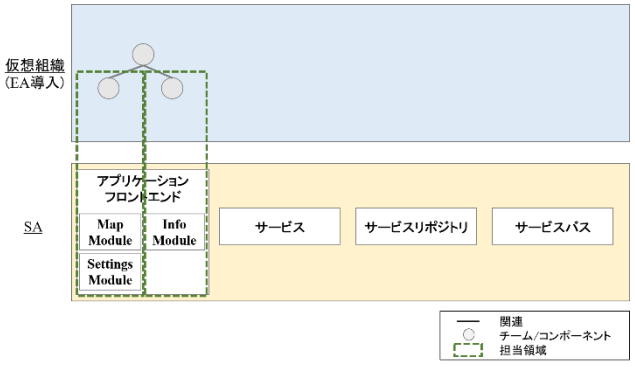

図 23 組織と SA（Clean Architecture）

本稿の評価作業はスマートフォンアプリケーション（アプリケーションフロントエンド）に対してドメインを考慮した SA である VIPER を適用していることから図 23 のケ

ースである．以上から SOA を EA 導入に適用することで組織構造と SA の関係を適切に繋ぐことが見込める．

10. 今後の課題

今後の課題は以下 2 点である．

(1) SOA を EA に適用：9.4 で組織と SOA について述べた．この点について SAAM for EA を用いた評価を行う必要がある．

(2) EA に適する SA の検討：9.4 で述べた組織と SOA の関係から EA に適する SA が持つ特性を検討する必要がある．

11. まとめ

EA 並行開発時のコンフリクト問題を着想の起点とし SAAM for EA を提案した．EA 開発フレームワークは開発プロセスのフレームワークであるので SA を考慮したフレームワークではない．本稿では EA 並行開発時のコンフリクト問題に対し EA 開発プロセスと SA の観点からアプローチすることで EA に適する SA 評価方法を提案した．

スマートフォン向けアプリケーションへ提案評価方法を適用しその有効性の評価を行なった．MVC と VIPER を比較した評価結果は実践事例の報告にある通りモノリシックな SA は EA 開発に適さないこと[14]を示した．この結果は EA 開発に適した SA 評価方法に必要な特性の一つは複数機能のインクリメンタル開発能力であることを示した．SAAM for EA は本稿提案の並行開発シミュレーション図を用い開発中の機能間の依存関係を時間軸と共に可視化するのでこの特性を有する．本稿は EA 開発組織と SA の関係に着目し EA 導入時に SOA を適用することを提案した．SOA を適用することで組織構造と SA の関係を適切に繋ぐことを考察した．

コンフリクトの原因となる開発中の機能間の依存関係はスケジュールの遅延という課題を発生させる．本稿提案の並行開発シミュレーション図は開発中の機能間の依存関係を時間軸と共に可視化することでスケジュール遅延を防ぐことが可能である．EA を導入する組織は並行開発シミュレーション図を用い複数のシナリオを検討することで最適なリリーススケジュールを選択しスケジュールの遅延を防ぐことができる．

参考文献

[1] K. Beck，Test Driven Development: By Example，Addison-Wesley Professional，2002.
[2] P. Clementsal, et. al., Evaluating Software Architectures, Addison Wesley, 2001.
[3] Digital.ai Software, 14th Annual State of Agile Report, 2020, https://stateofagile.com/#ufh-c-7027494-state-of-agile （参照 2021-04-26）.
[4] T. Dingsøyr, et al., Agile Development at Scale: The Next Frontier, IEEE Software, Vol. 36, No. 2, Mar./Apr. 2019, pp. 31-36.
[5] L. Dobricaal, et al., A Survey on Software Architecture Analysis Methods, IEEE Trans. on Software Engineering, Vol. 28, No. 7, Jul. 2002, pp. 638-653.
[6] E. Eric，Domain-Driven Design: Tackling Complexity in the Heart of Software，Addison-Wesley Professional，2003.
[7] N. Ford, et al., Building Evolutionary Architectures, O'Reilly Media, 2017.
[8] R. Knaster, and D. Leffingwell, SAFe 4.5 Distilled, Addison-Wesley Professional, 2018.
[9] D. Krafzig，et al，Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall, 2004.
[10] C. Larmana, et al., Large-Scale Scrum: More with LeSS, Addison Wesley, 2016.
[11] D. Leffingwell, SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises, Addison-Wesley Professional, 2018.
[12] R. C. Martin, Clean Architecture, Pearson, 2017.
[13] S. Newman, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2015.
[14] Uludağ, Ö., et al., Evolution of the Agile Scaling Frameworks, International Conference on Agile Software Development, pp. 123-139, Springer, 2021.
[15] J. M. Bass, and A. Haxby, Tailoring Product Ownership in Large-Scale Agile Projects: Managing Scale, Distance, and Governance, IEEE Software, Vol. 36, No. 2, Mar.-Apr. 2019, pp. 58-63.
[16] 田中 優之，青山 幹雄，複数プロダクトのエンタープライズアジャイル開発方法の提案と実践，情報処理学会 デジタルプラクティス，Vol. 11, No. 3, Jul. 2020, pp. 569-588.
[17] M. Tanaka , Kyoto Trip , https://apps.apple.com/at/app/id1516721339（参照 2021-01-04）.
[18] M. Tanaka , Kyoto Trip , https://github.com/masayuki5160/KyotoTrip（参照 2021-01--04）.
[19] Scaled Agile Inc , What's New in SAFe 5.0 , https://www.scaledagileframework.com/whats-new-in-safe-5-0/ （参照 2021-04-25）.