Vol.2021-SE-207 No.33
2021/3/2

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の実践事例が数多く報告されている．その実践には開発者間で同一ソースファイル更新の競合（コンフリクト）が発生により開発に遅延が生じることがある．プロダクトと開発組織をリードするマネジメント技術は，この課題に対処するための方法の一つであり，適切なソフトウェアアーキテクチャ設計もこの課題に対処する方法の一つである．本稿ではこの課題から着想した評価方法 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へ評価方法を適用する．評価結果から，SAAM for EA は提起した課題に対しソフトウェアアーキテクチャ及び開発プロセスのマネジメント観点からの評価に有効であり，エンタープライズアジャイル並行開発において発生する課題を事前に評価できる方法であることを確認した．このことから SAAM for EA は初めてエンタープライズアジャイルを導入する組織や事業の方針変更などによる影響を評価したい組織において有用な評価方法であると期待できる．

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の実践事例が数多く報告されている．その実践時には開発者間で同一ソースファイル更新の競合が発生すること（コンフリクト）により開発に遅延が生じることがある[7]．プロダクトと開発組織をリードするマネジメント技術はこの課題に対処するための方法の一つであり，適切なソフトウェアアーキテクチャ設計もこの課題に対処する方法の一つである．しかし Scaled Agile Framework (SAFe)[3][5], Large Scale Scrum (LeSS)，LeSS Huge[4]などの実践事例が報告されているエンタープライズアジャイル開発向けフレームワークでは，開発組織をリードするマネジメント技術は述べられるがソフトウェアアーキテクチャに関する議論は少ない．

本稿ではエンタープライズアジャイルとソフトウェアアーキテクチャの関係に着目し，エンタープライズアジャイルの並行開発時に発生する課題を事前に評価する方法を提案する．

2. 研究課題

本稿では，以下の 2 点を研究課題として設定する．

(1) エンタープライズアジャイル開発に適したソフトウェアアーキテクチャ評価方法の持つべき特性は何か

(2) エンタープライズアジャイル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 エンタープライズアジャイル

(1) Large Scale Scrum（LeSS）/ LeSS Huge

LeSS はスクラムをベースにしたエンタープライズアジャイル開発方法である[4]．図 1 に示すように，LeSS は複数のチームから構成される．各チームはスクラムチームとなっておりスクラムマスターが 1 名チームに所属する．製品およびサービスに責任を持つ担当者(プロダクトオーナ)が 1 名（開発者である必要はない）おり，製品に関する作業を集約したプロダクトバックログを管理しプロダクトの成長をリードする役割を担う．プロダクトバックログはスクラムにおけるそれと同様である．すべてのスクラムチームは同じスプリント期間で作業を行い，リリース可能な製品を開発する．各スクラムチームはスプリントプランニング, スプリントレトロスペクティブ(チームレトロスペクティブ)などスクラムのプラクティスを行う．また，複数のスクラム開発チームが同期を取りつつ開発を行うために各チームのスクラムマスターやプロダクトオーナなど役割を持つメンバが集まる打ち合わせ（セレモニ）がフレームワークのルールとして設定されている．

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

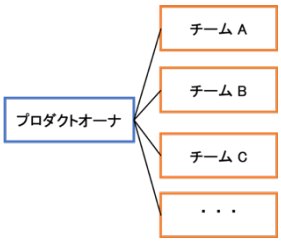

図 1 LeSS

チーム数が 8 以上の場合を対象とする開発方法が LeSS Huge である（図 2）.

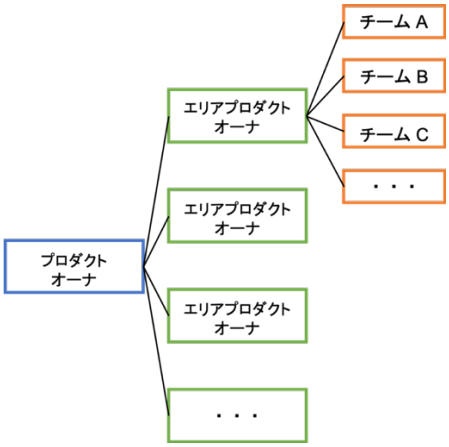

図 2 LeSS Huge

対象とするチーム数が 8 以上となっている理由は，これまでの実践事例から経験的に得られた数値である．LeSS Huge はエリアという概念を追加することにより LeSS よりも多くのチーム数で開発が可能となるよう設計されている[4]．エリアはプロダクトにおける一つの機能の開発を担当する．エリアプロダクトオーナはその機能の開発責任を負い，新たな機能の開発が必要なケースでは新しいエリアを追加することで組織を拡張する．プロダクトオーナは LeSS における場合と同様にプロダクトの成長をリードする役割を担う．

(2) **Scaled Agile Framework（SAFe）**

SAFe はエンタープライズアジャイル開発方法の一つである．SAFe の特徴は多くの開発チームでアジャイル開発が可能な方法を提案していることである[3][5]．また，SAFe は組織における投資や資産管理のマネジメント（ポートフォリオマネジメント）をフレームワークの概念として持っていることも特徴の一つである．エンタープライズアジャイルの事例としては最も多く報告されている一方で，複雑なフレームワークとなっており学習コストは少なくない．

SAFe が持つ特徴としてアーキテクチャ滑走路(Architectural Runway，以下 AR)がある．AR は SAFe においてソフトウェア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提案されている．組織が機能開発を優先し，ソフトウェアアーキテクチャの崩壊やその他技術課題の解決を後回しにすることがある．それはソフトウェアアーキテクチャの崩壊を招き，リスクとなる．AR はそれらを防ぐため，フィーチャチーム（機能開発チーム）とは独立して AR 開発チームを組織することでインクリメンタルな開発において継続的にソフトウェアアーキテクチャの維持と改善を推進する．

3.2 ソフトウェアアーキテクチャ

(1) **Clean Architecture**

階層アーキテクチャの課題であったドメイン層がインフラ層に依存する課題の解決策として提案がされたソフトウェアアーキテクチャの一つが Clean Architecture である．Clean Architecture は階層アーキテクチャに対して抽象レイヤを新たに追加することで外部ライブラリへの依存性を減らしビジネスロジックを変更に対して頑健にしている．ソフトウェアを階層に分割し，依存性のルールを守ることでソフトウェアテスト(ユニットテスト)を容易にするソフトウェアアーキテクチャパターンである[6]．図 3 に Clean Architecture の概念を示す．

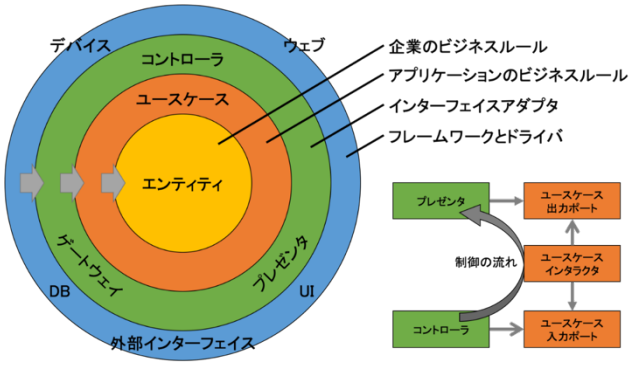

図 3 Clean Architecture

図 3 は依存関係が円の外側から内側へ向いていることを示している．階層アーキテクチャでは内側に存在したレイヤを，プロトコルを用いて抽象レイヤを利用することでこのアーキテクチャを実現している．Clean Architecture はプログラミング言語に依存しないソフトウェアアーキテクチャパターンであるが，実装言語の特性を考慮してそのソフトウェアアーキテクチャを実装することは容易ではない．しかし，後述の VIPER をはじめ Clean Architecture をベースに提案されているソフトウェアアーキテクチャは複数存在しており，レイヤの分割や依存性のルールという観点では同じ特性を持っている．

(2) **VIPER**

VIPER は Mutual Mobile 社が提案した Clean Architecture をベースにした iOS アプリケーション開発用のソフトウェアアーキテクチャである．図 4 に VIPER のソフトウェアアーキテクチャの構成を示す．Clean Architecture と同様に依存関係に向きがあるが iOS アプリケーション固有のモジュールとして画面遷移を担う Wireframe モジュールが追加されている．VIPER においても Clean Architecture と同様に

ソフトウェアをレイヤに分割し，依存性のルールを守ることでソフトウェアテストが容易となっている．

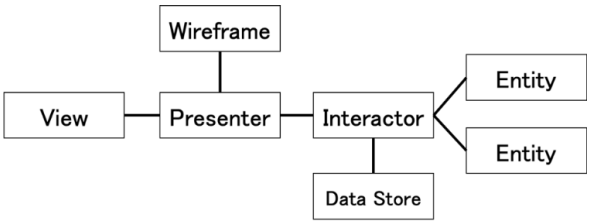

図 4 VIPER

3.3 Scenario-Based Architecture Analysis Method（SAAM）

SAAM は用意したシナリオに対してスコアリングを行うことで評価対象のソフトウェアアーキテクチャ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るため学習コストが低く，実施しやすいことも特徴の一つである．しかし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，本稿提案の評価方法をスマートフォンアプリケーションに適用し評価方法の評価を実施する．評価方法はソフトウェアアーキテクチャ観点とマネジメント観点の両方から組み立てた．提起した課題に対し，コンポーネントの粒度を小さくすること（ソフトウェアアーキテクチャ観点），そして並行開発時のリリースマネジメントを補助すること（マネジメント観点）が課題の解決に繋がるという仮説から評価方法は組み立てている（図 5）．

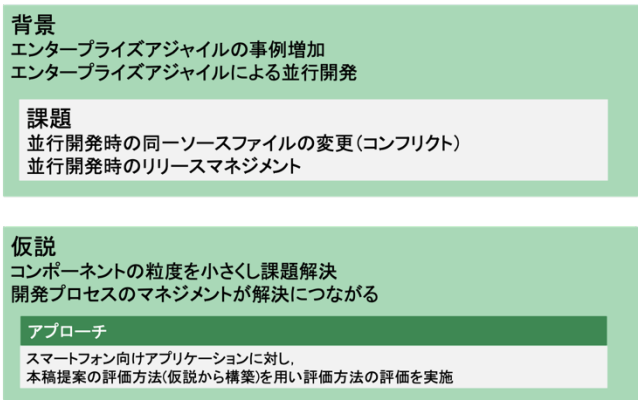

図 5 アプローチ

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

5.1 評価方法の概要

SAAM をベースとし，エンタープライズアジャイルにおいて課題となる並行開発時の評価を行うため SAAM for EA を提案する．SAAM for EA は SAAM において課題とされる[2]評価シナリオの準備方法と並行開発時に課題となるリリースマネジメント作業を可視化する方法を用意した．

5.2 評価シナリオの準備

評価シナリオは開発予定の機能リストを組み合わせることで並行開発を実施する評価シナリオを用意する．また，SAAM for EA においては評価シナリオを実施する組織を定義する．

5.3 並行開発シミュレーション図

SAAM for EA では本稿提案の並行開発シミュレーション図（図 6）を用いることでリリースマネジメント作業を可視化する．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．用意したシナリオごとに並行開発シミュレーション図を作成し，スプリントごとにリリースが問題なく実施できるかを評価する．

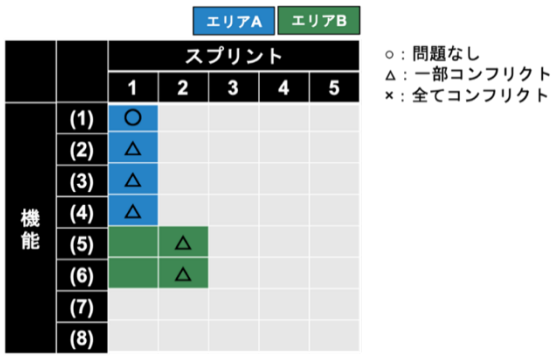

図 6 並行開発シミュレーション図

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各シナリオに対して作成する．

6. 評価対象ソフトウェア

6.1 機能

評価対象は著者が開発を行った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である[9]．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 ソフトウェアアーキテクチャ

MVC を用いたソフトウェアアーキテクチャを図 7，VIPER を用いたソフトウェアアーキテクチャを図 8 に示す．また，VIPER における地図に関するコンポーネントをまとめたコンポーネント群を図 9 に示す．VIPER においては図 8 に示すようにコンテキストが近いコンポーネント

をまとめており，地図の他のコンポーネント群についても図 9 で示す設計と同様の設計となっている[10].

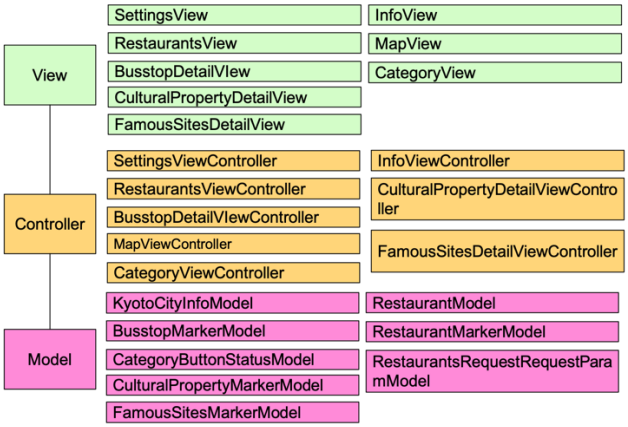

図 7 評価対象ソフトウェア（MVC）

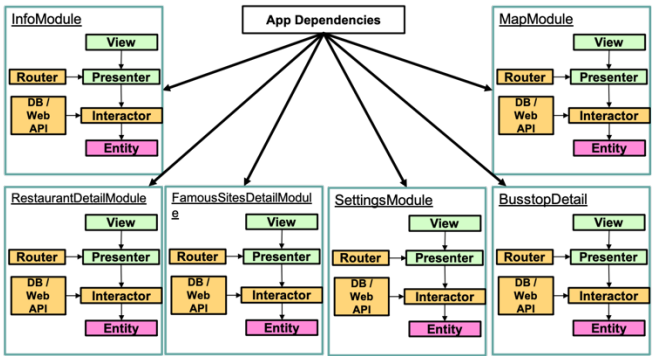

図 8 評価対象ソフトウェア（VIPER）

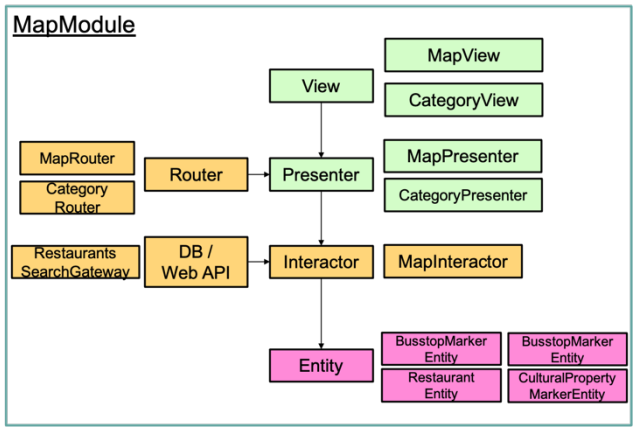

図 9 地図に関連するコンポーネント群（VIPER）

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．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．

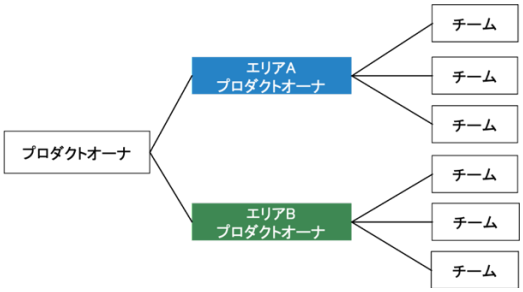

図 10 評価シナリオ（開発組織）

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

この時各シナリオの評価前の並行開発シミュレーション図は図 11 から図 15 のようになる．MVC と VIPER それぞれの場合において評価を実施する．

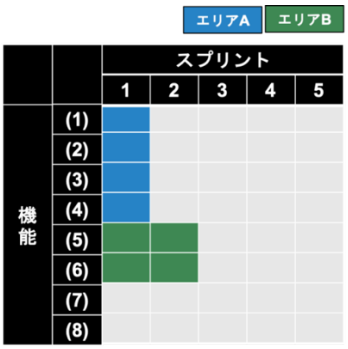

図 11 並行開発シミュレーション図（シナリオ 1）

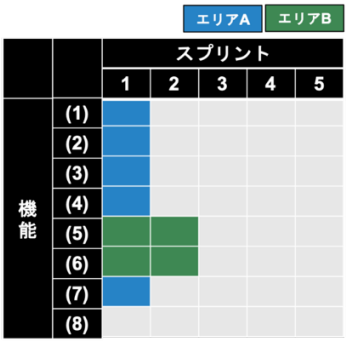

図 12 並行開発シミュレーション図（シナリオ 2）

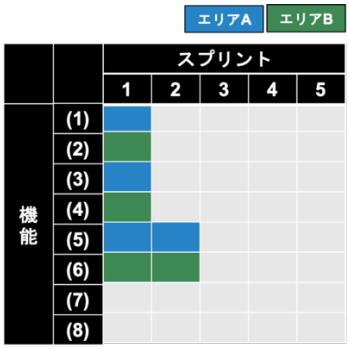

図 13 並行開発シミュレーション図（シナリオ 3）

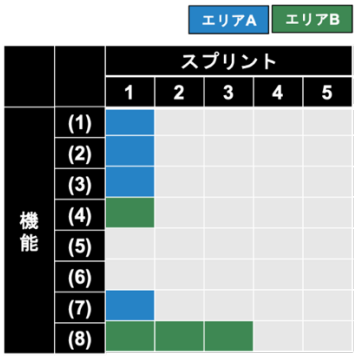

図 14 並行開発シミュレーション図（シナリオ 4）

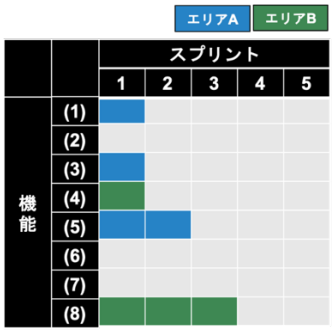

図 15 並行開発シミュレーション図（シナリオ 5）

8. 評価結果

8.1 ソフトウェアアーキテクチャ評価結果

表 4 に評価結果から作成した MVC と VIPER において各シナリオでコンフリクトしているクラスの評価とコンフリクトが発生するクラスをまとめた．各ソフトウェアアーキテクチャにおいて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．

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した．MVC においては MapViewController が変更対象のクラスになっている一方で VIPER では MapView と MapPresenter が変更対象となっている．これは VIPER のソフトウェアアーキテクチャの特性上，各コンポーネントの粒度が小さくなっていることによる結果である．このことから，VIPER では MVC の場合より各クラスでのコンフリクトの発生可能性が小さく，変更がしやすい可能性が考えられる．これはシナリオ 2 とシナリオ 3 においても同様のことが言えると考えられる．

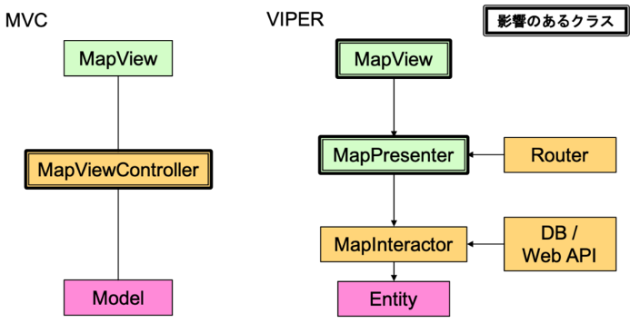

図 16 シナリオ 4 における比較

これらを踏まえて，評価結果を表 5 にまとめて示す．表中の+はそのソフトウェアアーキテクチャが変更に対して相対的に容易であることを示し，-は相対的に困難であることを示す．0 は差がないことを示す．

表に示すように 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．

表 5 ソフトウェアアーキテクチャ評価結果

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

8.2 並行開発シミュレーション図による評価結果

図 17 から図 21 に各シナリオでの MVC と VIPER の評価結果を示す．表中評価欄は○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．

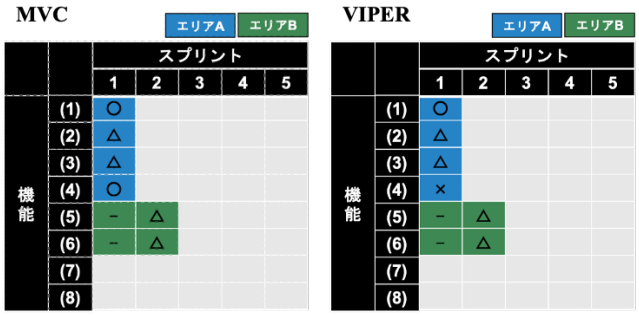

図 17 並行開発シミュレーション図結果（シナリオ 1）

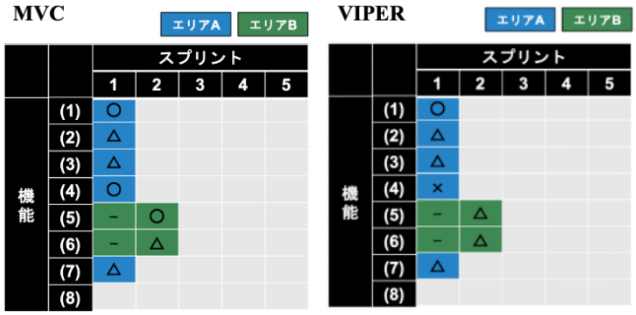

図 18 並行開発シミュレーション図結果（シナリオ 2）

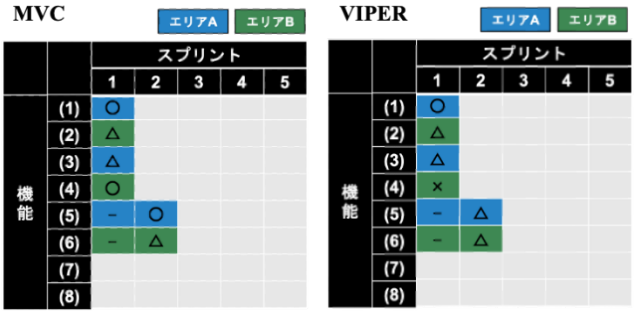

図 19 並行開発シミュレーション図結果（シナリオ 3）

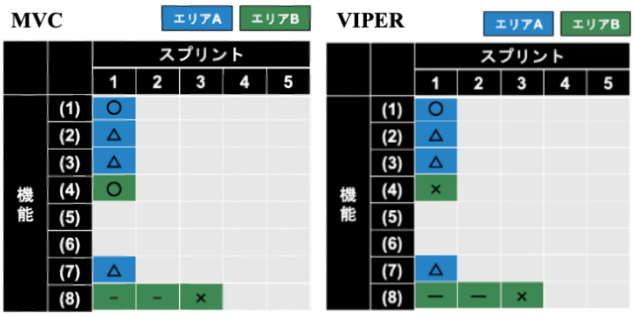

図 20 並行開発シミュレーション図結果（シナリオ 4）

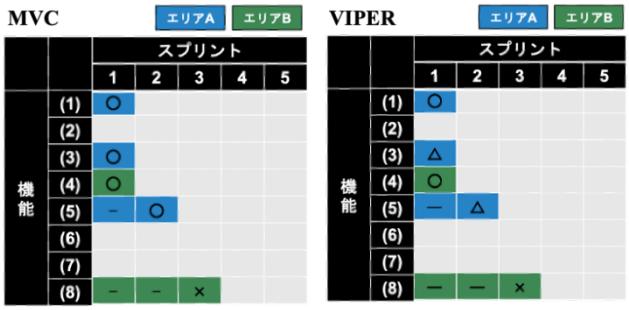

図 21 並行開発シミュレーション図結果（シナリオ 5）

MVC が VIPER と比較してコンフリクトを少なくリリースできるスプリントがあるシナリオを確認できる．図 22 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．

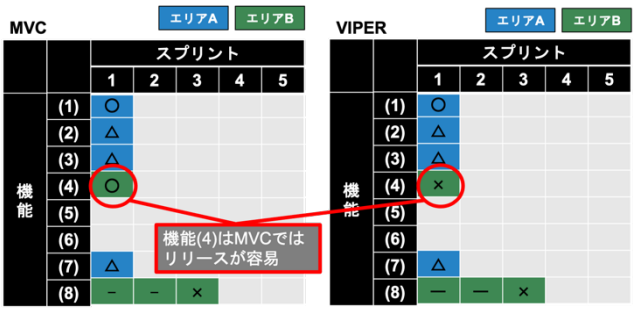

図 22 シナリオ 4 における並行開発シミュレーション図

シナリオ 4 のスプリント 1 において MVC が機能 4 のリリースが VIPER よりも容易である理由は図 23 で示すように変更範囲が MVC において Controller に集中していることが理由である．VIPER は各コンポーネントの粒度が小さく設計されているため複数のコンポーネントへの修正が発生しやすい．その結果がこの評価結果に表れている．

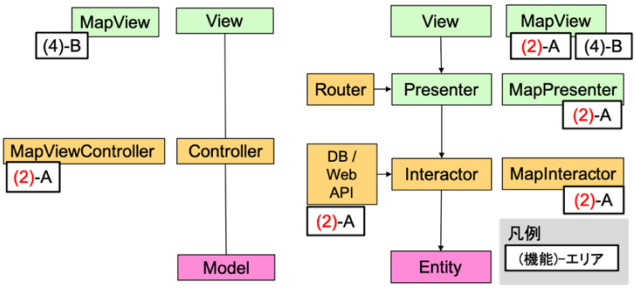

図 23 シナリオ 4 スプリント 1 における影響範囲の比較

他のシナリオにおいても同様に MVC の方がコンフリクトを少なくリリースできると評価されているケースが確認できる．その評価結果の差の原因はシナリオ 4 におけるケースと同様の理由からと考えられる．

9. 考察

9.1 研究課題（1）に対する考察

エンタープライズアジャイル並行開発時の課題から着想し SAAM for EA による評価を行なった（図 24）．評価結果から並行開発時の課題を実際に確認できた．並行開発シミュレーション図によるリリースマネジメント作業の評価はエンタープライズアジャイル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であると考えられる．

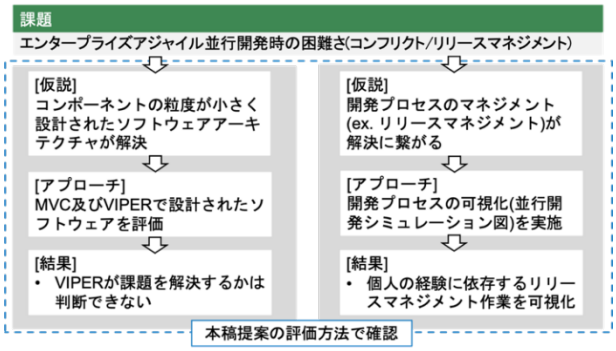

図 24 SAAM for EA による評価結果

9.2 研究課題（2）に対する考察

本稿で提案した並行開発シミュレーション図は開発を予定している複数の機能を現在のリソースとリリース調整を考慮してスケジュールを組み立てるマネジメント作業を可視化する図となっている．これはエンタープライズアジャイルにおいて組織とプロダクトの開発計画の状況を見てマネジメントを担うメンバが担当する責務を可視化していると考えられる．図 25 にエンタープライズアジャイルのプロダクトマネージャが担当する役割を示す[8]．図中 Groom Prioritizer ではプロダクト（ソフトウェア）に対する要求を把握した上で，並行開発シミュレーション図で可視化したように最適な開発シナリオをシミュレーションすることで判断をしていると考えられる．個人の経験に依存すると考えられるこの作業を並行開発シミュレーション図によって可視化できると考える．

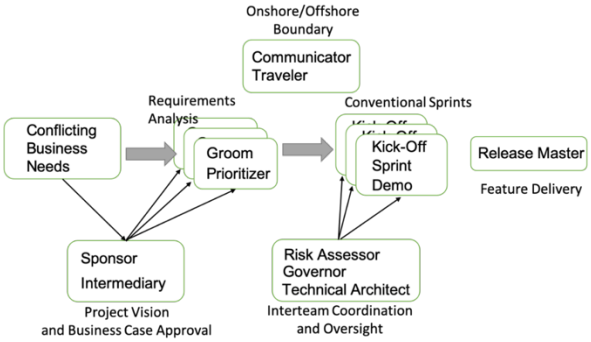

図 25 エンタープライズアジャイルのプロダクトオーナ

9.3 ソフトウェアアーキテクチャと開発組織

ソフトウェアアーキテクチャがプロセスに制約を及ぼすことを考えると，ソフトウェアアーキテクチャが開発組織に反映されることが望ましい（図 26）．しかし事業の方針変更や組織変更など様々な理由が原因となりソフトウェアアーキテクチャが開発組織に適切に反映されないことが発生する．本稿で提案した SAAM for EA を適用することでそのような場合に発生する開発への影響を事前にシミュレーションできることが期待できる．

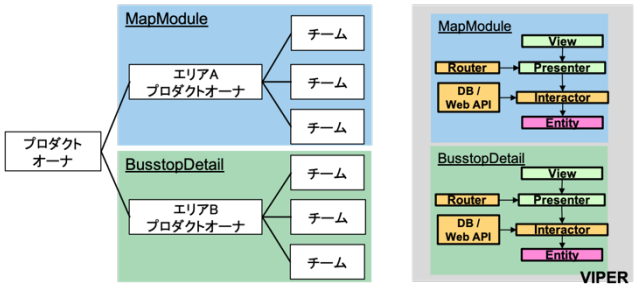

図 26 ソフトウェアアーキテクチャと開発組織の関係

9.4 SAFe における AR 開発チームの必要性

SAFe において AR 開発チームはプロダクトの開発を担うチームと独立して技術負債への対処やソフトウェアアーキテクチャの維持，改善に取り組む．SAAM for EA を実施した結果確認できた課題点については AR 開発チームのように継続的にソフトウェアアーキテクチャの維持と改善を推進するチームを設けることで対応をすることが必要と考えられる．

SAFe は他のエンタープライズアジャイルのフレームワークと比較して大規模な組織への適用を可能としている．そのため並行開発時に発生する課題を見込んだルール設定となっていると考えられる．AR 開発チームはその課題に対処するために設定された概念の一つであり，その必要性は本稿の評価結果からも確認ができた．

9.5 エンタープライズアジャイルフレームワークの選択

エンタープライズアジャイルは組織の拡大に対応できる必要がある．LeSS はより大規模な組織に対応するために LeSS Huge をフレームワークとして提供し，SAFe は小規模

な組織向けの Essential SAFe からより大規模な組織向けの Full SAFe を提供している．組織はそれぞれのフレームワークの特徴を把握した上で数あるエンタープライズアジャイルのフレームワークを選択する必要があり，それは困難な作業である．フレームワークを選択するために検証の必要性が出てくると考えられるが SAAM for EA はエンタープライズアジャイルにおいて課題となる並行開発時の問題について評価が可能であり，組織がエンタープライズアジャイルのフレームワークを選択する際の事前検証に有用であると期待できる．

10. 今後の課題

今後の課題は以下 2 点である．

(1) 他システムでの SAAM for EA を用いた評価実施
(2) SAAM for EA が持つべき他の特性の検討

他のシステム，評価シナリオを用いて SAAM for EA の評価を実践することを進める必要がある．また，エンタープライズアジャイルにおける課題を調査することで SAAM for EA が持つべき特性が他に必要か検討する必要がある．

11. まとめ

本稿で提案した評価方法 SAAM for EA をスマートフォン向けアプリケーションへ適用し，その有効性の評価を行なった．MVC と VIPER に対しソフトウェアアーキテクチャの評価を実施した結果，コンポーネントの粒度が小さく設計されている VIPER が MVC よりもいくつかの評価シナリオにおいて変更が容易であるという結果を得た．しかし，VIPER が課題を解決するソフトウェアアーキテクチャとは判断できないことも確認した．

また，エンタープライズアジャイル（LeSS または LeSS Huge）において有用と考えられる開発プロセスの可視化を補助する方法として並行開発シミュレーション図を提案した．本稿で提案した SAAM for EA はこれからエンタープライズアジャイルを導入する組織や組織変更や事業の方針変更による影響をシミュレーションする方法として有用と期待できる．そのため他のソフトウェアやシナリオを用いた評価を実施することで SAAM for EA の有効性を確認し，今後の課題とし提起した課題に対して更に研究を進める．

参考文献

[1] P. Clementsal, et. al., Evaluating Software Architectures, Addison Wesley, 2001.
[2] L. Dobricaal, et., A Survey on Software Architecture Analysis Methods, IEEE Trans. on Software Engineering, Vol. 28, No. 7, Jul. 2002, pp. 638-653.
[3] R. Knaster, and D. Leffingwell, SAFe 4.5 Distilled, Addison-Wesley Professional, 2018.
[4] C. Larmana, et al., Large-Scale Scrum: More with LeSS, Addison Wesley, 2016.
[5] D. Leffingwell, SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises, Addison-Wesley Professional, 2018.
[6] R. C. Martin, Clean Architecture, Pearson, 2017.
[7] 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.
[8] 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.
[9] M. Tanaka , Kyoto Trip , https://apps.apple.com/at/app/id1516721339（参照 2021-01-04）.
[10] M. Tanaka , Kyoto Trip , https://github.com/masayuki5160/KyotoTrip（参照 2021-01--04）.