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

M2019SE008　田中 優之
指導教員　青山 幹雄

1 研究背景

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．

2 研究課題

本稿では，研究背景を踏まえ以下の 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補助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

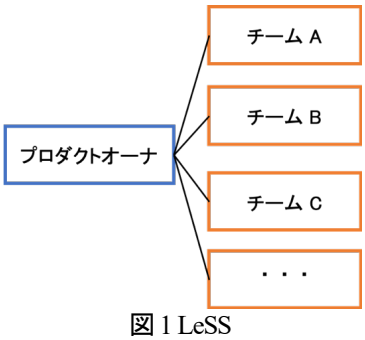

図 1 LeSS

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．製品およびサービスに責任を持つ担当者（プロダクトオーナー）が 1 名（開発者である必要はない），製品に関する作業を集約したプロダクトバックログを 1 つだけ持つ．この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．すべての開発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．

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)．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．

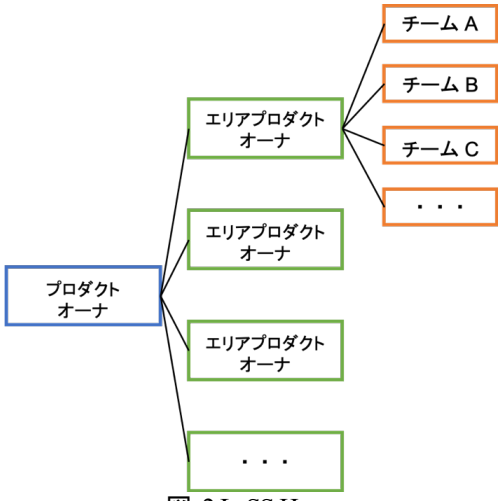

図 2 LeSS Huge

(2) Scaled Agile Frameowrk（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．

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強くしている．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．

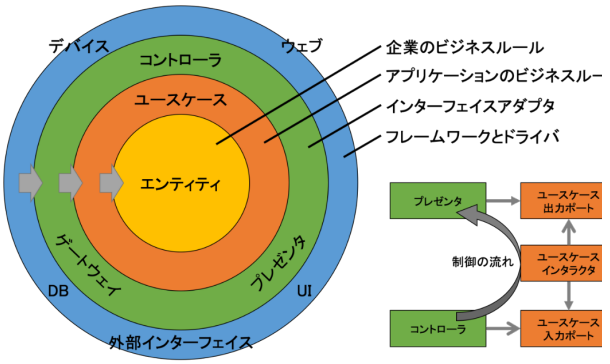

図 3 Clean Architecture

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．

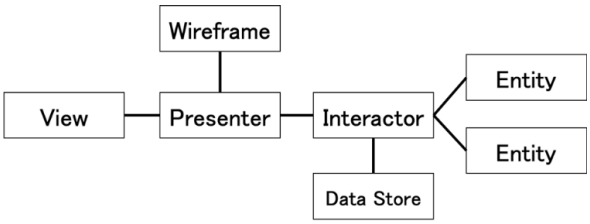

図 4 VIPER

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．しかし網羅的に品質特性を評価する方法としては ATAM が適している．

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 に本稿のアプローチを示す．

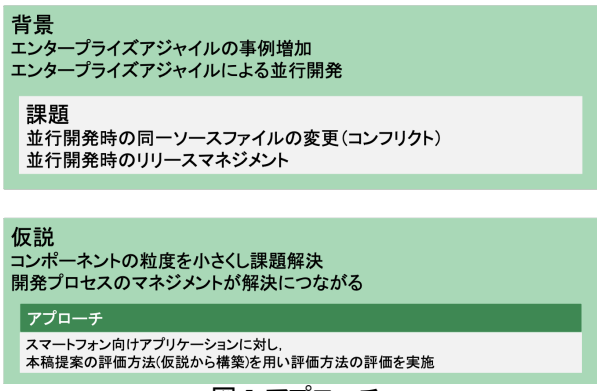

図 5 アプローチ

5 評価対象ソフトウェア

5.1 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

5.2 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 6，VIPER を用いたソフトウェアアーキテクチャを図 7 に示す．公開しているスマートフォンアプリケーションは VIPER を用いて実装をしている[10]ため MVC については それを参考に設計を行った．

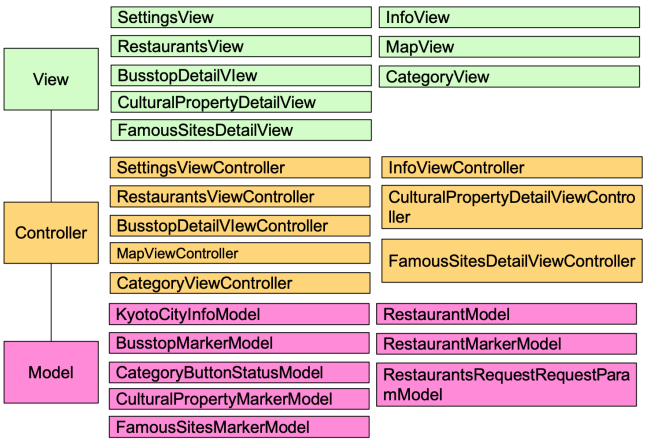

図 6 評価対象ソフトウェアアーキテクチャ(MVC)

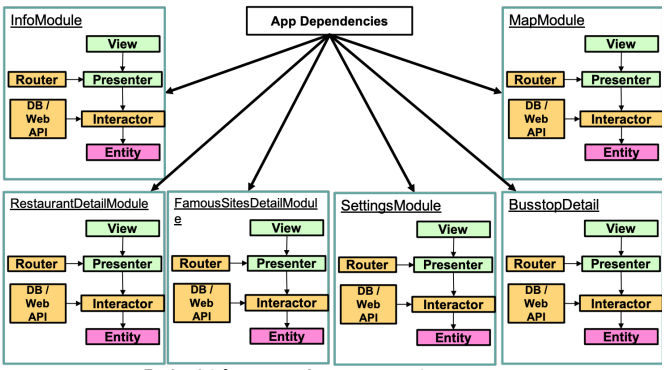

図 7 評価対象ソフトウェアアーキテクチャ(VIPER)

6 ソフトウェアアーキテクチャ評価方法

6.1 評価シナリオ

LeSS Huge を実践する開発組織(図 8)において複数の機能を並行で開発するシナリオを用いることで，MVC と VIPER の場合について SAAM を用いたソフトウェアアーキテクチャ評価を実施する．

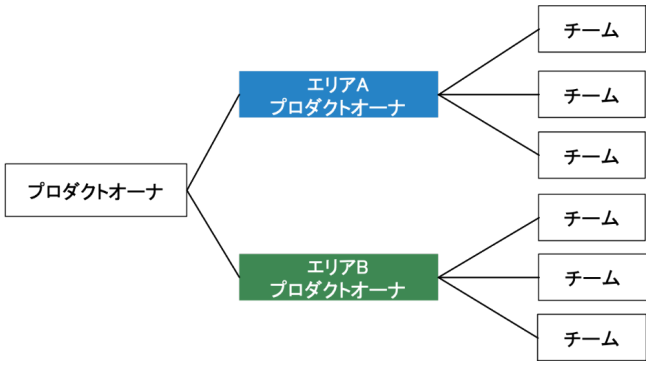

図 8 評価シナリオ(開発組織)

表 2 に開発を予定している機能の一覧およびその開発予定期間を示す．なお 1 スプリントは 2 週間として開発期間を算出した．表 3 に本稿の評価シナリオを示す．表 2 で示した機能を図 8 で示した開発組織で並行開発するシナリオとした．

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

6.2 評価

SAAM を用いた MVC および VIPER のソフトウェアアーキテクチャの比較を表 3 に示したシナリオで実施する．また，各スプリントでの開発状況を時系列で評価を行う．図 9 に本稿で提案する並行開発シミュレーション図を示す．図中○はリリース時のコンフリクトが発生しないことを示し，△は一部コンフリクトが発生することを示す．×は全てのクラスファイルでコンフリクトが発生することを示す．複数の機能を並行開発する際に各エリアでの作業進捗の影響が問題になることがあり[7]，それらを可視化する．

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

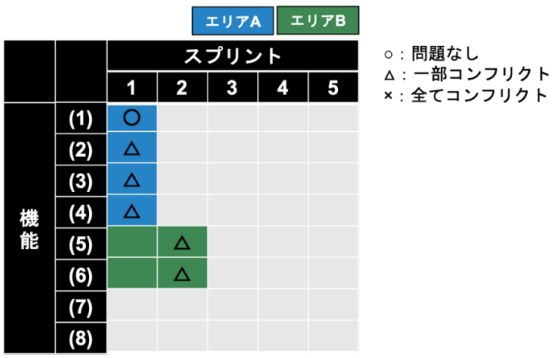

図 9 並行開発シミュレーションの実行

7 評価結果

7.1 MVC と VIPER の評価結果比較

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○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 10 にシナリオ 4 における各クラスの変更範囲を示した．MVC においては MapViewController が改修対象のクラスになっている一方で VIPER では MapView と MapPresenter が改修対象となっている．これは VIPER のソフトウェアア

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

ーキテクチャの特性上，各クラスが持つ役割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度合いが小さく対処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．

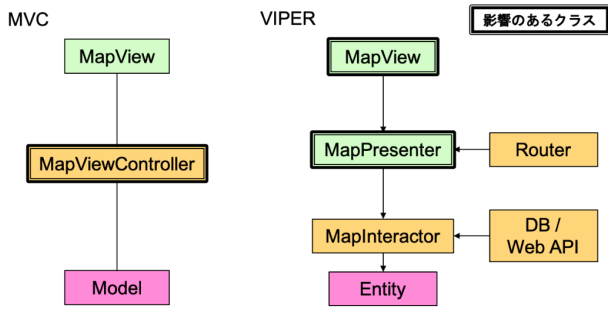

図 10 シナリオ 4 における比較

これらを踏まえると評価結果を表 5 にまとめられる．表中の+はそのソフトウェアアーキテクチャが相対的に良いことを示し，-は相対的に悪いことを示す．0 は差がないことを示す．表で示すように 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．

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

7.2 エンタープライズアジャイル開発のリリースマネジメント

本稿提案の並行開発シミュレーション図を用いて各シナリオの開発プロセスとリリース時のコンフリクト状況の可視化を行ったが大きな差は確認できなかった．

8 考察

エンタープライズアジャイル における課題（複数機能の並行開発の困難さ）はソフトウェアアーキテクチャ，プロジェクトのマネジメントの両面でのアプローチが必要と考えられる．並行開発シミュレーション図は個人の経験に依存するリリースマネジメント作業を可視化することで，提起している課題への解決に繋がると考える(図 11)．

ソフトウェアアーキテクチャがプロセスを示すことを考えるとソフトウェアアーキテクチャは開発組織に反映されることが望ましい．しかし，事業の方針変更や組織変更，開発の優先度変更など様々な理由でそうならないことは発生する．本稿提案の評価方法はそれら状況の変化が起こる際に実施することで発生する事象を事前にシミュレートできると考える．また，これからエンタープライズアジャイルを導入する組織が導入により発生する事象を事前にシミュレートできることも期待できる．

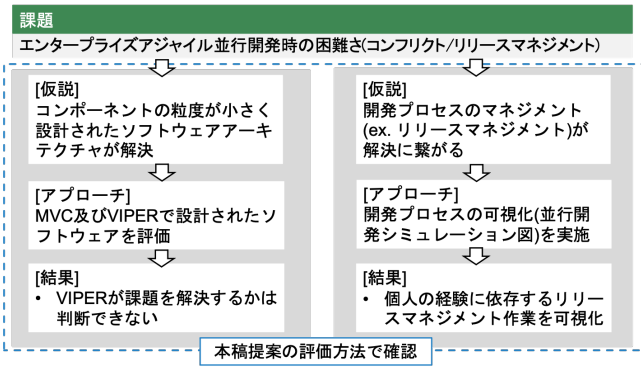

図 11 評価結果に対する考察

9 今後の課題

(1) 別システムを用いた提案評価方法を用いた評価の実施
(2) SPLE (Software Product Line Engineering)および ASD (Agile Product Line Engineering) の観点からの考察

10 まとめ

本稿において提起した課題はソフトウェアアーキテクチャだけでなくマネジメント面からのアプローチも同時に必要であると考えられる．本稿提案の評価方法はエンタープライズアジャイルを導入する組織がその導入のシミュレーションを行う方法として有用と期待できる．

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, 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, 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] Masayuki Tanaka , Kyoto Trip , https://apps.apple.com/at/app/id1516721339（参照 2021-01-04）.
[10] Masayuki Tanaka , Kyoto Trip , https://github.com/masayuki5160/KyotoTrip（参照 2021-01--04）.