リリース頻度を向上したアジャイル開発への段階的移行に関する一考察

研究コース3　ソフトウェアテスト

研究員

上野 彩子（個人）

山口 信二郎（NTT Communications）

主査

喜多 義弘（長崎県立大学）

副主査

上田 和樹（日本ナレッジ（株））

背景

こんな経験ありませんか？

背景

我が社では従来ウォーターフォールで開発してきたが、世の中はアジャイルだな…。市場ニーズに適合したプロダクトができるらしい。
我が社でもアジャイル(スクラム)を導入するぞ！

ユーザーストーリーにポイント付けて、
スプリントプランニングやって、振り返りやって…。
よし、うまくやれてるな！

あれ？　ウォーターフォールに「スプリント」って区切りを付けただけになっている？？
リリース頻度もウォーターフォールの時と変わらない…！？

背景

そもそもアジャイルって何のためにやるんだっけ？？

流行ってるから導入しただけになっちゃったかも…💦

アジャイルの目的を整理

アジャイルの目的（の1つ）**は、**

「市場のニーズを捉え、ビジネス価値の高いプロダクトを提供する」

と表現できる。

目的達成の為のアプローチは、

・フィードバックを元に、プロダクトの方向性を随時軌道修正していく。

（市場ニーズにマッチしたものを開発する為）

・リリース頻度を上げる。

（フィードバックをQuickに集める＆フィードバックを反映したものをリリースする為）

つまり「**小まめにフィードバックを集め、小まめにリリースする**」と言うアプローチ。

アジャイルの効果

アプローチはわかったけど、ほんとに効果あるの？？

リリース頻度が高い企業の方がパフォーマンスがよいという定量データがある！[1]

エリート企業は、低パフォーマンス企業の**973**倍の頻度でリリースしている！

（他にもリードタイム、MTTR、変更失敗率の指標でも差がある.）

アジャイルをするなら、

リリース頻度上げてフィードバックを貰わないと意味がない！

[1] States of DevOps 2021

しかし！！

リリース頻度の高い理想的なアジャイル

の実践って**なぜか**できていない…。

理想的なアジャイル実践の障壁の考察

組織や社内ルールに起因する問題

・出荷判定会議の承認が得られない限りリリースが出来ず、リリース頻度を上げづらい。

・ジョブローテーションなどでチームメンバーが固定されず、スクラムチームが成熟しづらい。

・組織が職能別に分かれている。

例えばQAチームと開発チームが明確に分かれており、ウォーターフォールのようにテストが後工程となってしまい、シフトレフトがしづらい。

個人の技術不足の問題

・TDD等の、アジャイルに適した開発技術を知らない / 習得していない。

上記のQAチームと同様、シフトレフトしない為、手戻りコストが大きくなる。

理想的なアジャイル実践の障壁の考察

アジャイルのコンセプト理解不足（冒頭のやりとりはコレ）

・そもそもリリース頻度とフィードバックが重要だと認識していない。

アジャイルの形式（スプリントに区切る、振り返りイベントをやる…etc）を真似するだけで、本来の目的を意識した取り組みができていない。

・製造業ベースの品質保証の考え方をしてしまっている。

変更を前提とせず、リリース時には完璧な厳しい品質管理をクリアしたものだけをリリースしようとする。 → テストを重視しすぎ、リリーススピード低下。

…etc

問題点のまとめ

確かにうちのアジャイル、ウォーターフォール感が否めない…。似非アジャイルかも…。
他にも色々と間違ったやり方してる部分がありそう…。

リリース頻度が重要なのはわかったけど、リリース頻度を上げる具体的な方法がわからない…。

いきなり超イケてるアジャイルを目指すのは
ハードル高そう…。

研究テーマ

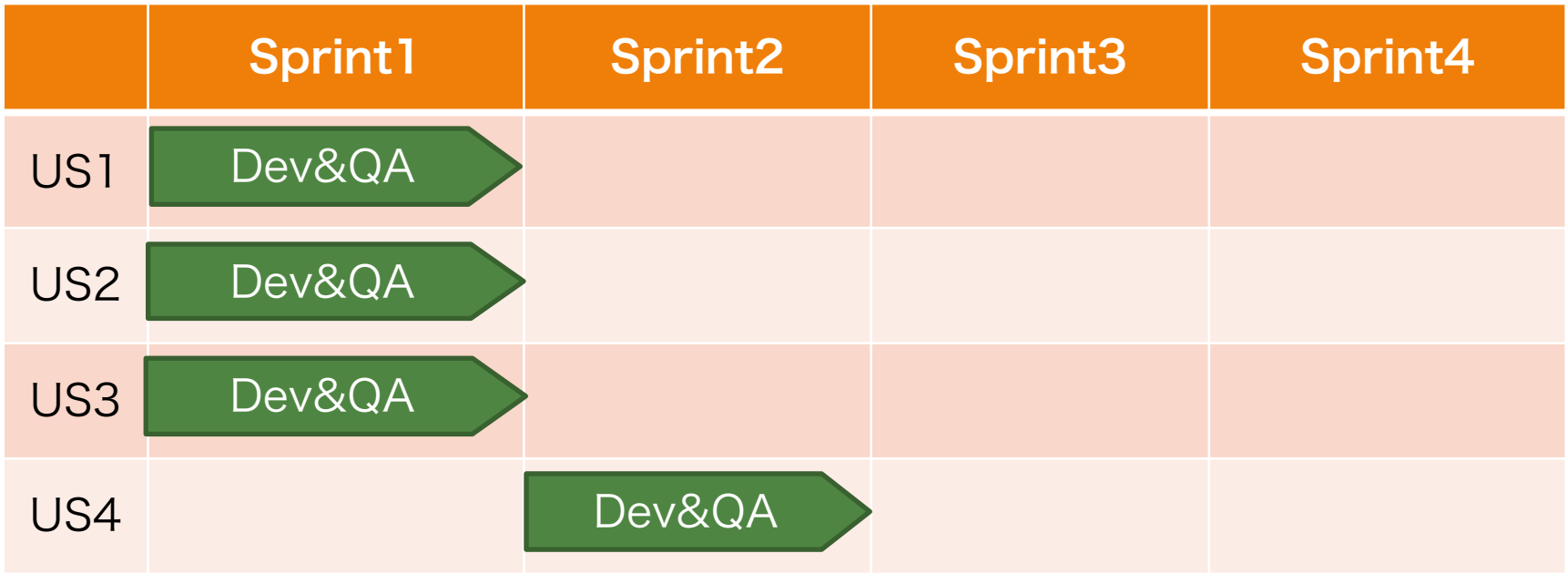
・ウォーターフォールからアジャイルに移行した組織が陥りがちな
「似非アジャイルパターン」を定義し、**問題を明確化**する。

・いきなり完璧なアジャイル（理想アジャイル）を目指すのが難しい為、
「中間解」を定義し、それを目指すことで似非アジャイルから**脱却する方法を提示**する。

＊理想アジャイルほどのリリース頻度とはならないが、似非アジャイルよりはリリース頻度が上がる手法となる。
＊前に挙げた3種類の問題（組織問題、技術問題、コンセプト理解問題）の内、主として
技術問題、コンセプト理解問題を解決する事を狙う。

本論文で定義する理想的なアジャイル

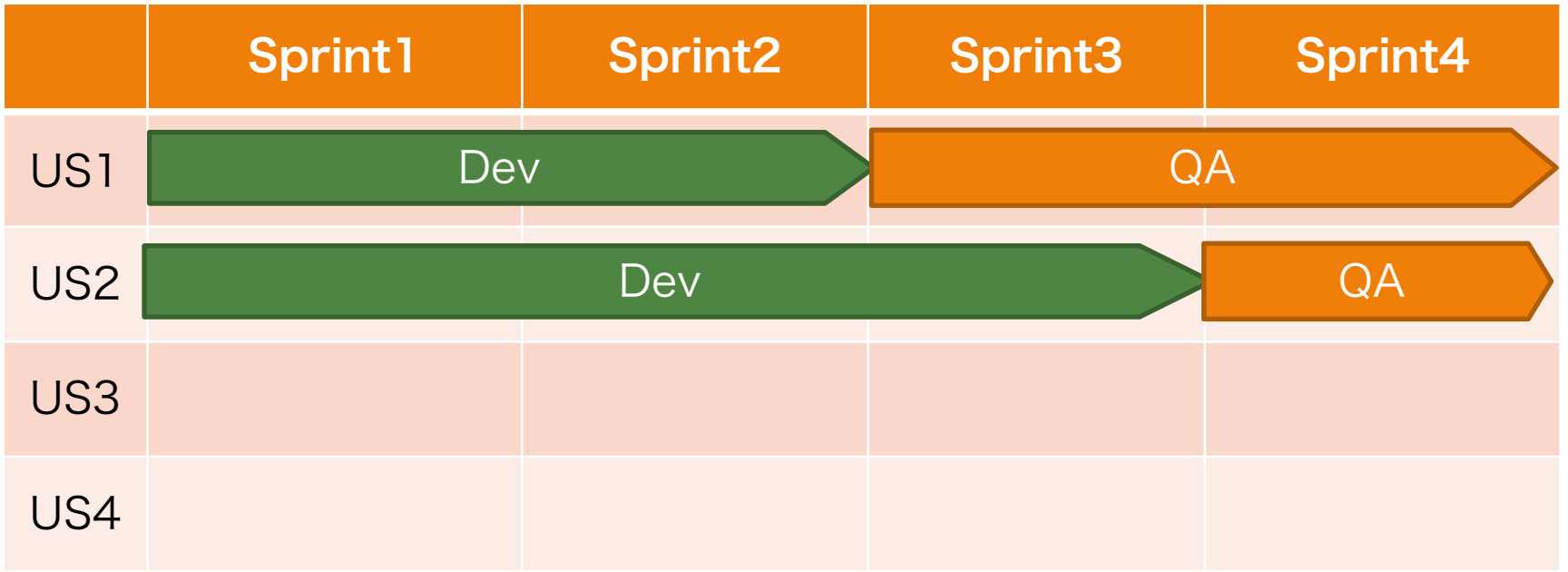
スプリントの終わりにリリース可能な成果物ができあがっているため、
ユーザーからのフィードバックに基づいて**ビジネス価値を追求**できる



- ユーザーストーリーは一つの顧客体験を含むような大きさに分割される
- 開発とQAを同時に行っていく
- ユーザーストーリーごとに必要なテストを選択し，順不同で行う
- 開発とQAはスプリント内で完結し，リリース可能な状態となる

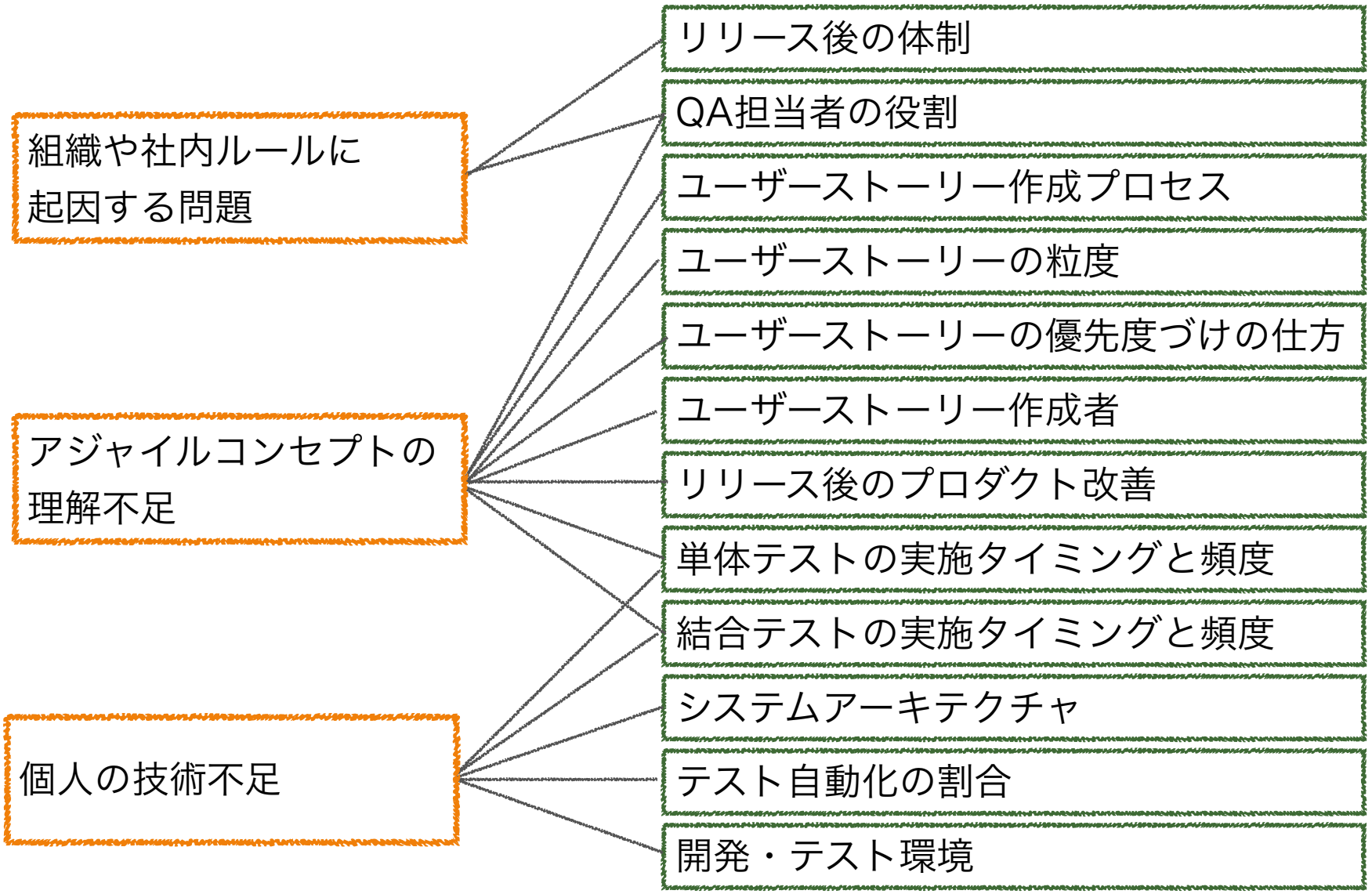
本論文で定義する似非アジャイル

- ユーザーストーリーの完成までに時間を要し，**リリース頻度が低下**する
- エンドユーザーからの**フィードバックを受ける機会が減少**し，アジャイルのメリットを享受できない



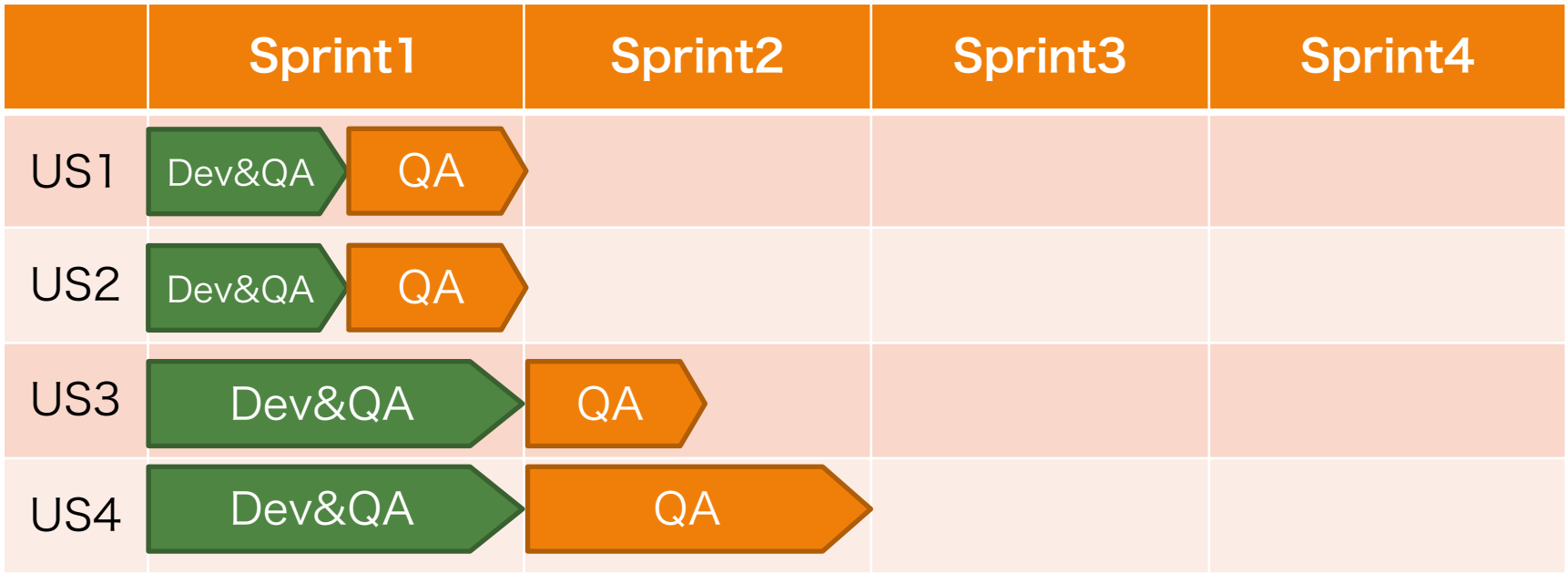
- 1つのユーザーストーリーのサイズが大きく，1スプリントで完成しない
- 開発とQAが完全に分かれている
- テストはQAの期間内に各テストレベルを順に実行する
- 多くのテストが自動化されておらず，リグレッションテストに工数がかかる

リリース頻度が上がらない原因



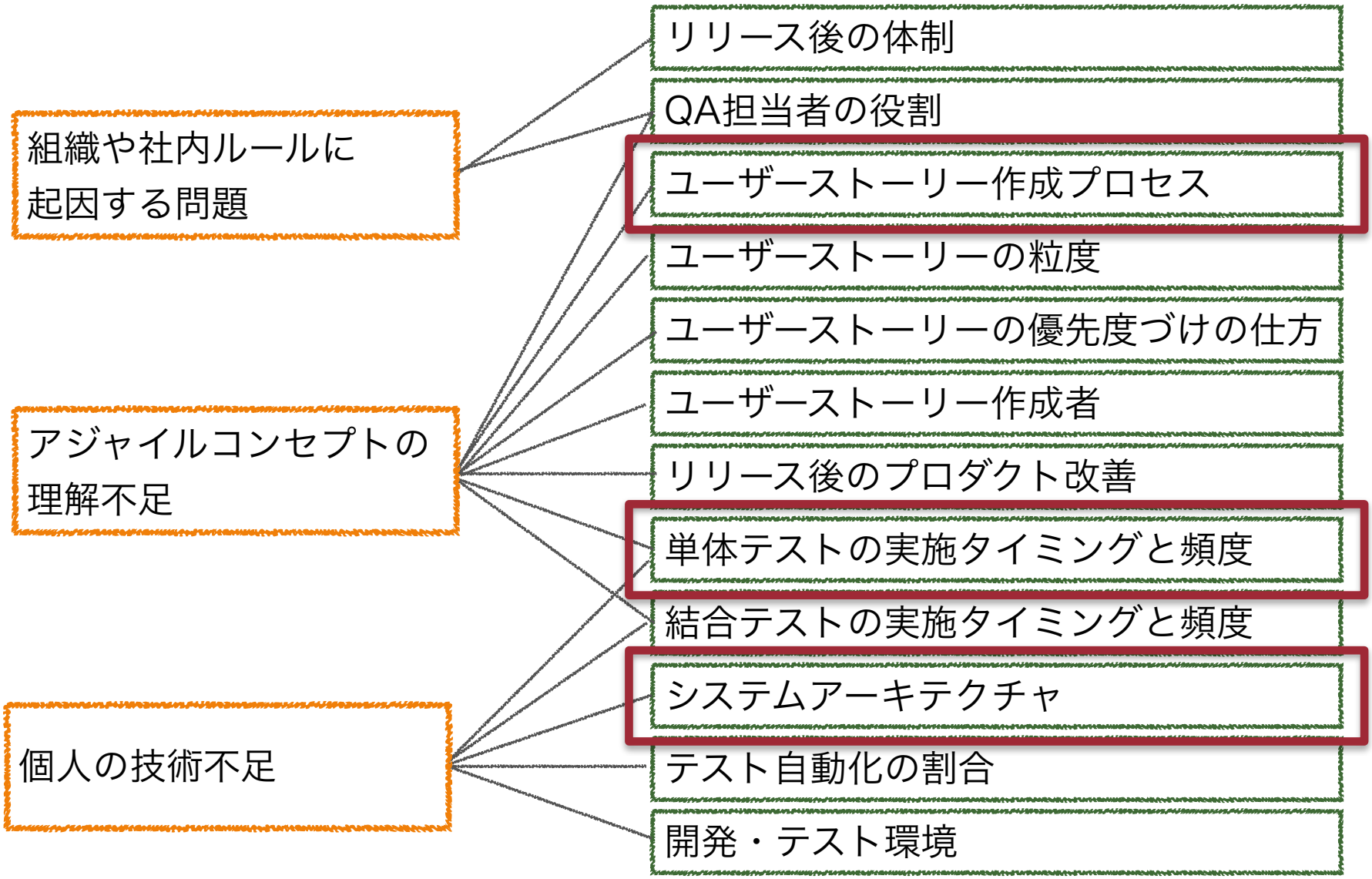
中間解の提案

・移行の初期段階にありがちな似非アジャイルを正した中間解を定義する
・すぐには変えにくい**組織課題を回避**しながら，**リリース頻度をあげる**取り組みを行う



・開発と一部のQAを同時に行う
・似非アジャイルよりも1スプリントで完了するユーザーストーリーの数が多い

リリース頻度が上がらない原因



中間解の特徴 - 単体テスト実施タイミングと頻度

似非アジャイル	理想アジャイル	中間解
実施しない	開発中に何度も行う	開発中に何度も行う

不具合の修正コストは発見が遅れるほど指数関数的に増大するため，開発と同時に自動化された単体テストを実行し，手戻りを防ぎリリース頻度の向上を目指す.
デベロッパーが不慣れな場合，QA 担当者がデベロッパーを育成したり，テスト環境を整備したりするなど，テスト活動をリードすることが望ましい.

中間解の特徴 - システムアーキテクチャ

似非アジャイル	理想アジャイル	中間解
複数機能を束ねる巨大な塊	コンポーネントの独立性が高く，開発，テスト，デプロイ，保守，運用がしやすい	コンポーネントの独立性が比較的高い

複数のデベロッパーが並行してそれぞれ小さなユーザーストーリーを開発することが前提となるため，コードをマージした際のコンフリクトを回避し，小さな単位でのテストを容易にすると，リリース頻度の向上につながる

中間解の特徴 -ユーザーストーリーの作成プロセス

似非アジャイル	理想アジャイル	中間解
スプリント開始前に 一気に作る	スプリントごとの フィードバックに 基づき継続的に作成	フィードバックに 基づき継続的に作成. 取得の頻度は低い

完全なフィードバックサイクルを組むことは目指さないものの，フィードバックを促しながら継続的なプロダクト改善を行う.
ユーザーストーリー作成にはデベロッパーも出来るだけ参加し，チーム全体でビジネス価値を意識したプロダクト開発を行うこと目指す.

中間解アジャイルの使い方

”ユーザーストーリーの完成にかかる期間”や”リリース頻度”の目標を立てる

達成できている

現行プロセスはよさそうだ.
次の目標を立てよう！

達成できていない

ユーザーストーリーのサイズが肥大しているかも

開発品質に問題が発生してリリース頻度が下がっているかも

指標を管理することで継続的にチームのアジャイル手法のどこに問題があるかを考えるきっかけとなる

中間解アジャイルを使うとこんないいことがあるはず

中間解アジャイルを目指せば，似非アジャイルよりリリース頻度を高められそう！

中間解アジャイルではフィードバックに着目しているので，似非アジャイルに比べて市場のニーズに適合できそう！

中間解アジャイルを実践しリリース頻度が上がれば，組織を変えていく取り組みも経営層に説明しやすそう！

制約事項

理想的なアジャイル開発へ移行するためには，**組織の意思決定プロセスや社内ルール，契約などの制約**に対応したほかの指標を取り入れることが必要であると考えられる.

中間解から理想的なアジャイル開発への移行については論じていない.

結論

ウォーターフォールからアジャイルに移行した組織が陥りがちな
「似非アジャイルパターン」を定義し、**問題を明確化した.**

似非アジャイルから徐々に理想的なアジャイル開発に近づけていくための
一つの**中間解を提案した**.

中間解を目指すことにより，**リリース頻度を徐々に高めながら市場からの早い
フィードバックを取り入れる**開発ができる可能性について考察した.

理想的なアジャイルを目指すには**組織の意思決定プロセスや社内ルールなどの観点**
も併せて考察する必要があり，この点は今後の課題である.