Agile Japan 2019 セッション資料

その常識、変えてみせる。

SHIFT

分業による高速開発を実現

上流から下流まで、品質保証で
アジャイル開発を支援するSHIFTの取り組み

2019／07／18

自己紹介

技術推進部 アジャイルファンクション

谷岡　俊祐

「アジャイルを紐解いて見えてきた
QAサクセスフロー事例紹介」

技術推進部 アジャイルファンクション

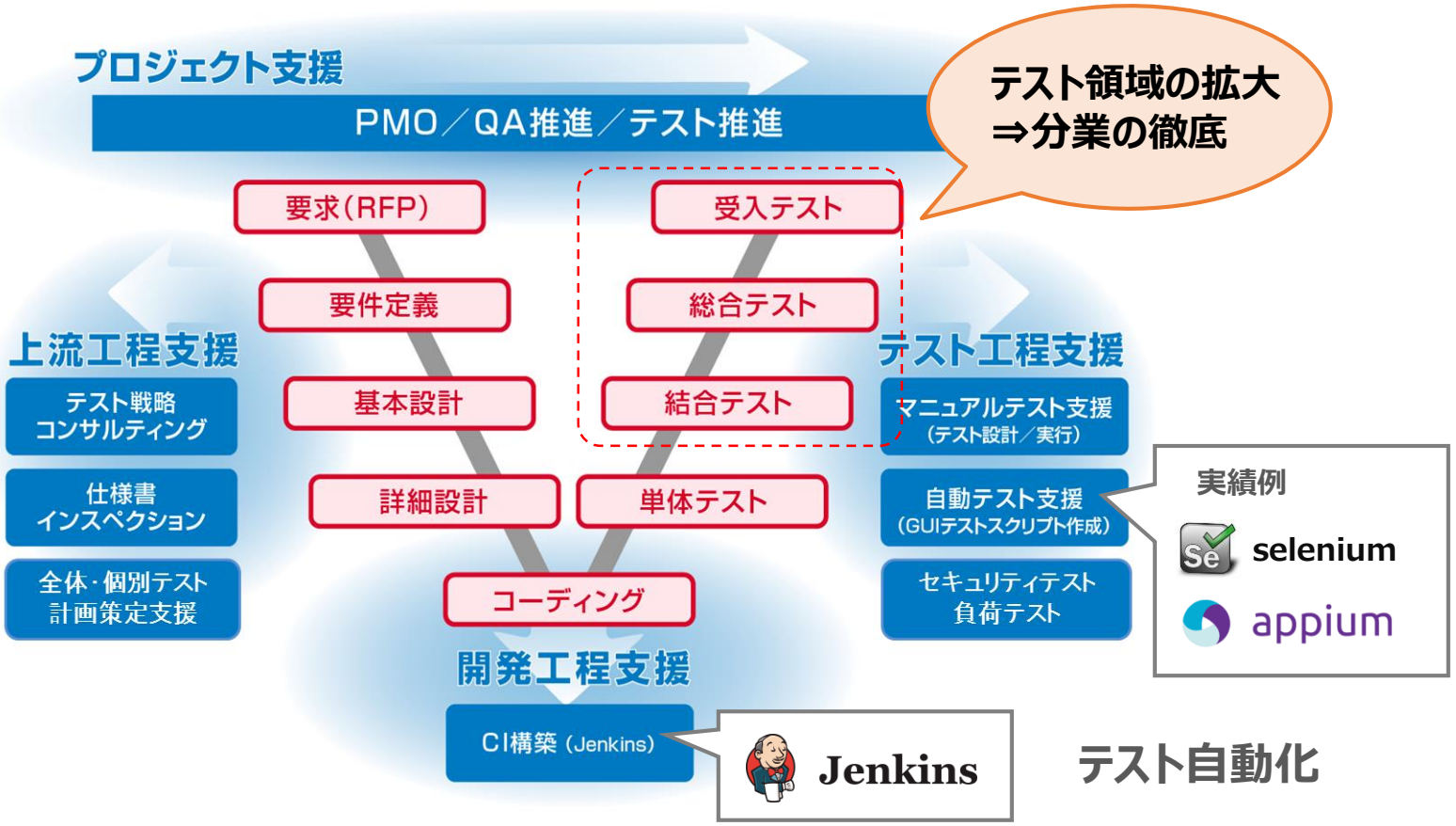
上野　彩子

「大規模なチーム体制で目指すAgile」

提供サービス

ソフトウェアが正しく動作することを**定義づけ　確認する**
ソフトウェアの中に必ず存在する**不具合を見つける**

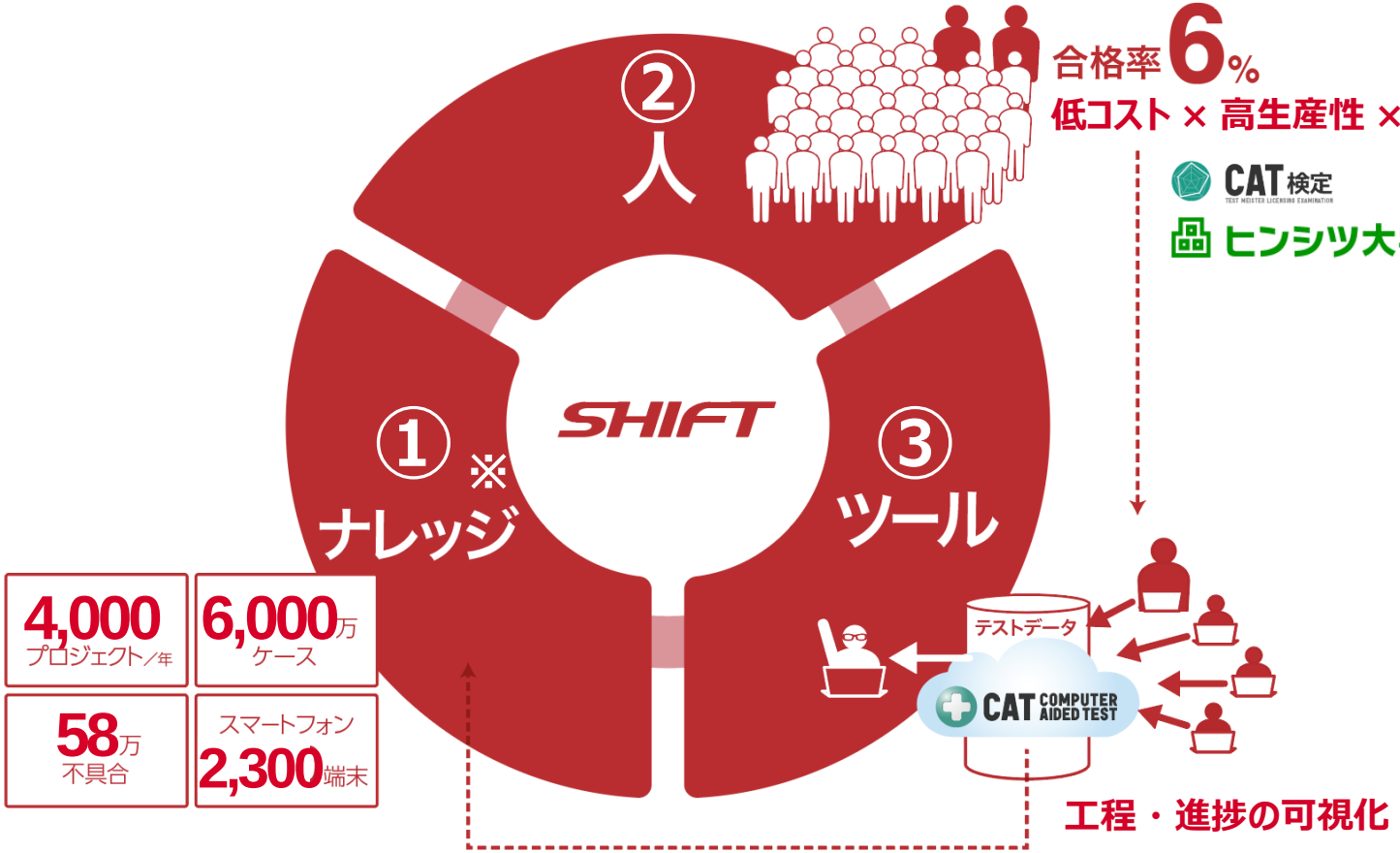
<テストサービスの全体像>



SHIFTの強み

高い品質のサービスが提供できる SHIFTにしかない3つの強み

①ナレッジ（実績）　②人（生産性）　③ツール（可視化）



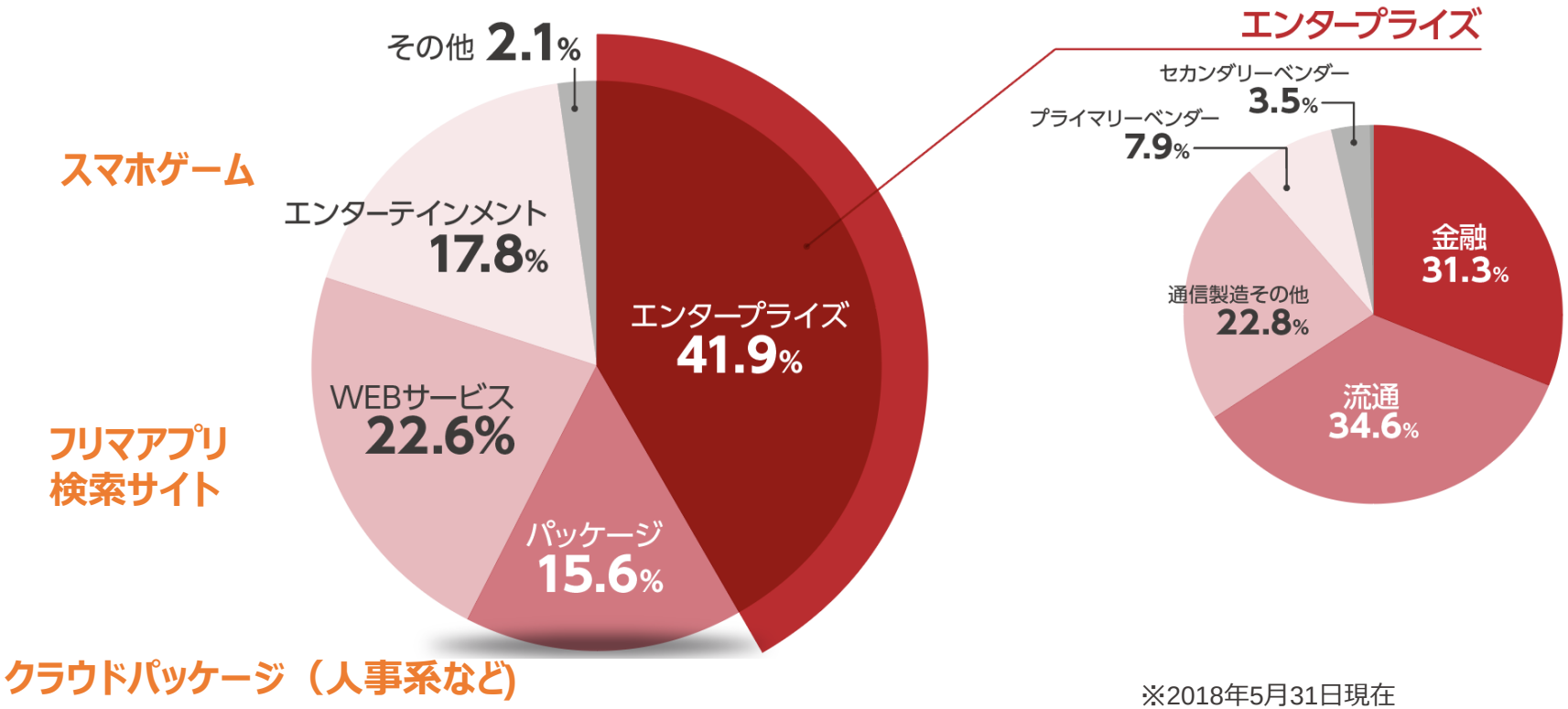
※2018年10月末時点　当社調べ（端数を調整しています）

SHIFTの強み　①ナレッジ（実績）

品質保証の支援実績　約12,000プロジェクト　1,000社　2,300製品
近年はエンターテイメント・WEBサービス系の依頼が増加

金融業・流通業といったエンタープライズ系企業を主軸とし、
スマホゲーム等エンターテイメント系まで幅広い企業とのお取引きがあります。

売上高に占める各領域の割合



※2018年5月31日現在

アジャイルを紐解いて見えてきた QAサクセスフロー事例紹介

自己紹介

谷岡　俊祐

(たにおか　しゅんすけ)

品質・技術統轄部

技術推進部

アジャイルファンクション

- SIerにて自治体向けシステムに対する上流開発からテスト、保守運用までを幅広く経験。
- SHIFT入社後、ECサイトのGUIテスト自動化や、CI環境構築、分散メッセージシステム構築などの自動化基盤構築を担当する。
- 現在はアジャイル開発におけるQAチーム体制構築および継続的なリリースに対し、システムの安定化を目指したサービス提供を行なっている。

AGENDA

1. はじめに
2. アジャイル現場の悩み
3. 解決へ向けたアプローチ
4. 品質担保に向けたサクセスフロー
5. 各フローにおける対策と結果
6. まとめ

アジャイルの解釈

12原則について、
「継続的により良い解決方法を探している状態」
私はこれをアジャイルと解釈します。原則は以下ですが、
解法のベストプラクティスはなく現場とチームで作り上げます。

====== ====== ====== ======アジャイル宣言の背後にある原則 ====== ====== ====== ======

私たちは以下の原則に従う:

1. **顧客満足を最優先**し、価値のあるソフトウェアを早く継続的に提供します。
2. 要求の変更はたとえ開発の後期であっても歓迎します。**変化を味方につける**ことによって、お客様の競争力を引き上げます。
3. 動くソフトウェアを、2-3週間から2-3ヶ月という **できるだけ短い時間間隔でリリース**します。
4. ビジネス側の人と開発者は、プロジェクトを通して **日々一緒**に働かなければなりません。
5. **意欲に満ちた人々**を集めてプロジェクトを構成します。環境と支援を与え仕事が無事終わるまで彼らを**信頼**します。
6. 情報を伝えるもっとも効率的で効果的な方法は **フェイス・トゥ・フェイス**で話をすることです。
7. **動くソフトウェア**こそが**進捗**の最も重要な尺度です。
8. アジャイル・プロセスは持続可能な開発を促進します**。一定のペースを継続的に維持**できるようにしなければなりません。
9. **技術的卓越性と優れた設計**に対する 不断の注意が機敏さを高めます。
10. **シンプル**さ（ムダなく作れる量を最大限にすること）が本質です。
11. 最良のアーキテクチャ・要求・設計は、**自己組織的なチーム**から生み出されます。
12. チームがもっと効率を高めることができるかを**定期的に振り返り**、それに基づいて自分たちのやり方を最適に調整します。

====== ====== ====== ====== ====== ====== ====== ====== ====== ======

[参考文献]『**アジャイル開発宣言**』
http://agilemanifesto.org/iso/ja/principles.html

アジャイルの解釈

12原則について、
「継続的により良い解決方法を探してい
私はこれをアジャイルと解釈します**が、**
解法のベストプラクティス**げます。**

計画、テスト、品質!?

私たちは以下の原則に従
1. **顧客満足を**
2. 要求の変更は**き上げます。**
3. 動くソフトウェア**を**
4. ビジネス側の人と
5. **意欲に満ちた人々**　**無事終わるまで彼らを信頼**します。
6. 情報を伝えるもっと**で話をすることです。**
7. **動くソフトウェア**こそ
8. アジャイル・プロセスは**。一定のペースを継続的に維持**できるようにしなければなりません。
9. **技術的卓越性と優れた**　**不断の注意が機敏さを高めます。**
10. **シンプル**さ（ムダなく作れ**量を最大限にすること）**が本質です。
11. 最良のアーキテクチャ・要求・設計は、**自己組織的なチーム**から生み出されます。
12. チームがもっと効率を高めることができるかを**定期的に振り返り**、それに基づいて自分たちのやり方を最適に調整します。

[参考文献]『**アジャイル開発宣言**』
http://agilemanifesto.org/iso/ja/principles.html

その１. 顧客満足を優先してリクエストを叶えていくんだ！

- 特定のユーザしか使わない機能ばかりが増え結局使いづらい
- スケーラビリティが制限されたシステムになる
- バグ報告やリクエストの収集がつかない。開発ボリュームが多すぎる

その2. 変化を許容し短納期で定期サイクル回すんだ！

- 変更許容を頻繁に行うことで仕様刈り取りや共有が困難
  →品質低下のトリガー
- 時間がないので開発はするが整理ができずに属人化
- 作りかけたドキュメントが増え、横串で仕様を把握する人がいなくなる

その3. 自己組織的に高い技術で動くものを作れば進捗だ！

- 仕様策定が曖昧なまま開発者に委ねた結果、想定外の成果物が出てくる
- 個人に委ねた開発が続くと組み合わせた結果、画面間差異が生まれもする
- 開発力が高いチームほど、コミュニケーションコストの方が高くなる

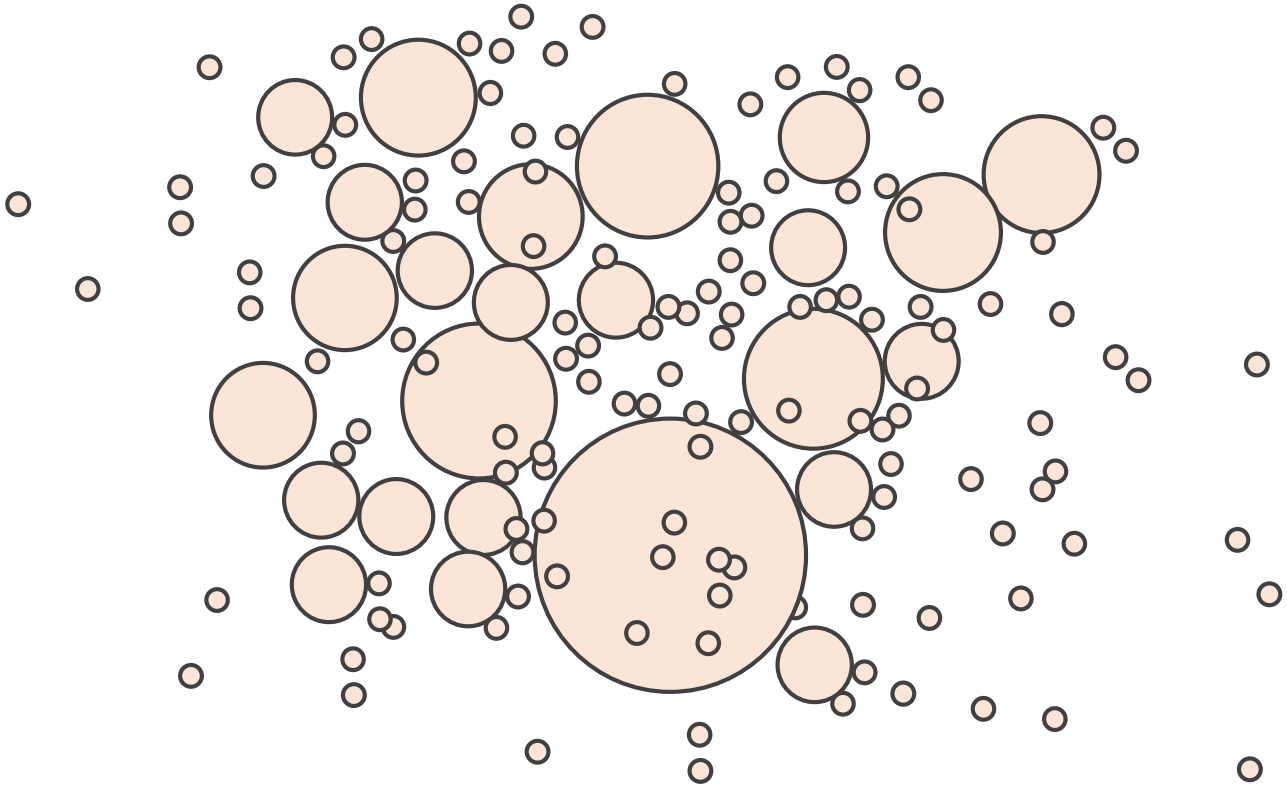
アジャイル現場の悩み

1. 顧客満足を優先してリクエストを叶えていくんだ
   - 特定のユーザしか使わない機能ばかりが増え結局
   - スケーラビリティが制限されたシステムに
   - バグ報告やリクエストの収集が
2. 変化を許
   - 難→品質低下のトリガー
   - 属人化
   - て仕様を把握する人がいなくなる
3. に高い技術で動くものを作れば進捗だ！
   - 様策定が曖昧なまま開発者に委ねた結果、想定外の成果物が出てくる
   - 個人に委ねた開発が続くと組み合わせた結果、画面間差異が生まれもする
   - 開発力が高いチームほど、コミュニケーションコストの方が高くなる

総合解釈の必要＋品質も担保
→総合プロデュースでリードタイム短縮

現場のリクエスト

要求／指摘／決定／発言／期待 … の分散と集合



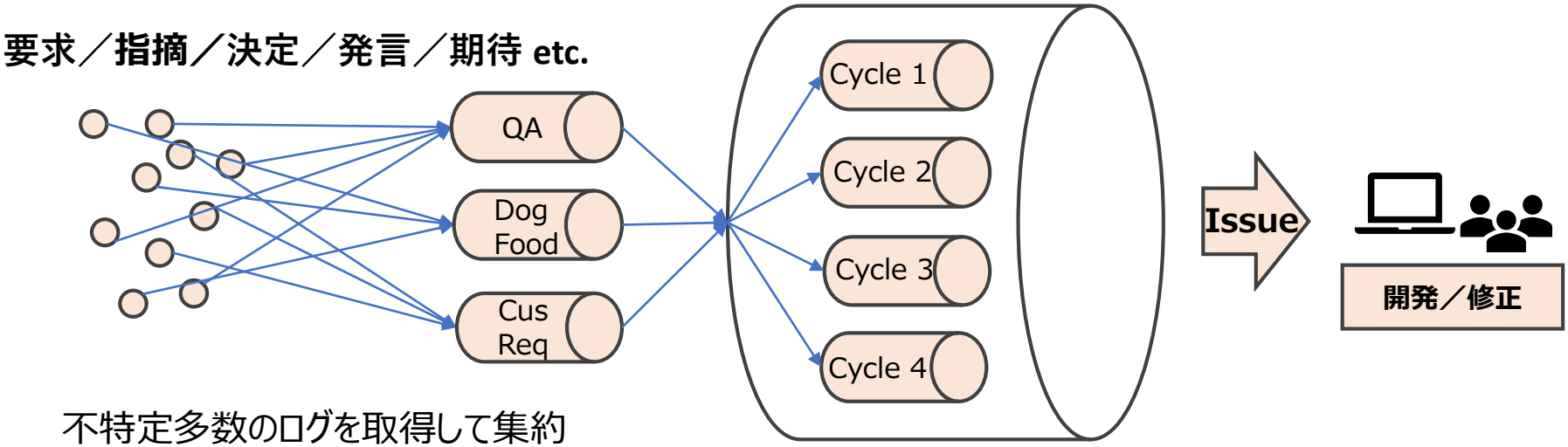
粒度・優先度
期日・大きさ
はさまざま

From End User/Sales/PM/PO/AnotherDivision/Dev/ QA…

共通アプローチ

■適切な集約管理

- 書き出しましょう（取得）
- 同じ場所で管理しましょう（集約）
- 分けて並べましょう（分別）
- 優先度をつけましょう（優先）
- 誰もがいつでも簡単に見れる状態にしましょう（活用） → 現場安定



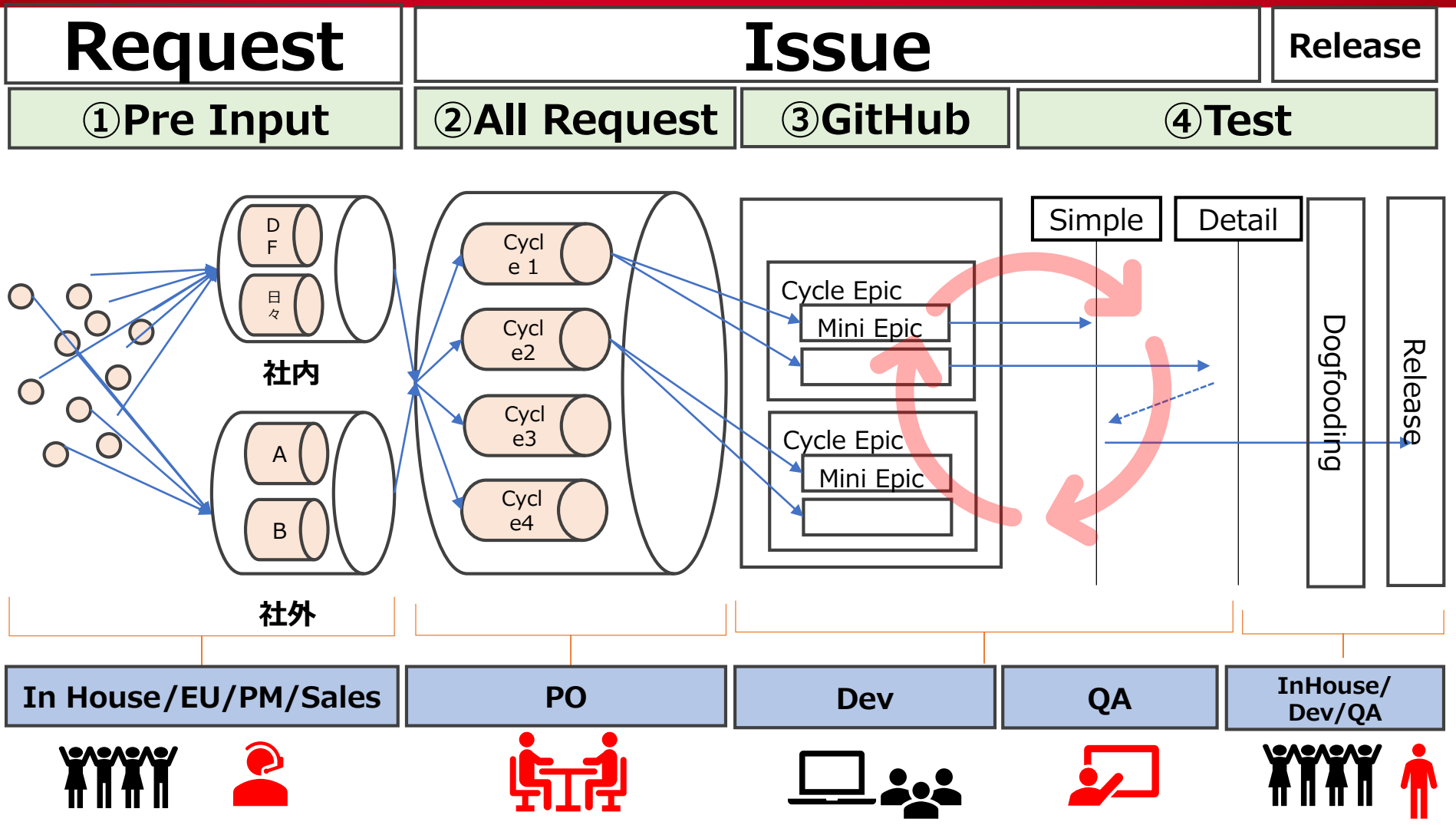
不特定多数のログを取得して集約
分別キューに詰めて、優先度を決め再集約
適切な場所に配布し活用する
※キュー＝資料：知見、傾向分析

解決に向けたアプローチ

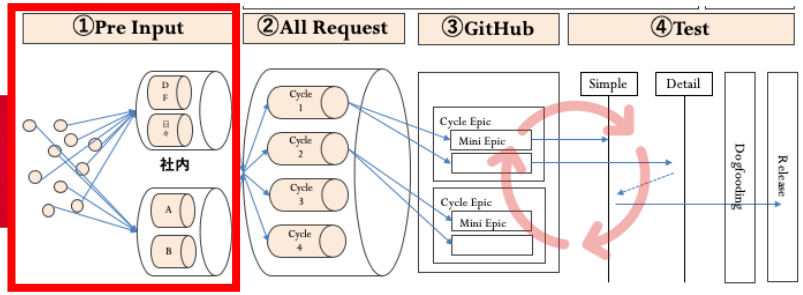
①リクエストを適切に管理
↓
②要件背景／事業方針／利益を加味して
　優先度付バックログを整理
↓
③Issue内に明確な仕様策定を行いDevと壁打ち
↓
④簡易テスト→詳細テストを実施、FBも③と同じ場所へ
↓
リリース

※スプリントプランニング、デイリースクラム、各種レビュー、レトロスペクティブももちろん実施

品質担保に向けたサクセスフロー



5.フローでの対策と結果



①リクエストを適切に管理(Pre Input)

[対策]

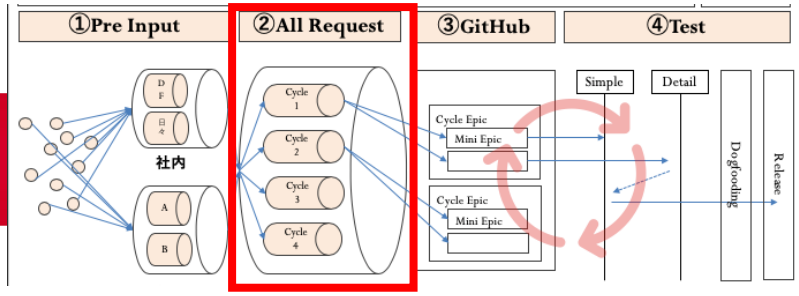
- 社内：気づきは全て載るように発見者に委任方式とし間口を広く浅くとる。
- 社外：窓口を一本化。アンケート形式で背景や必要項目を必須へ。
- 社内社外から優先度の高いものを集約

[結果]

- リクエスト集約コストが低下
- 漏れなく刈り取れるので取りこぼしや記載漏れの低下

リードタイム短縮！

5.フローでの対策と結果



②要件背景/事業方針/利益から優先付とバックログ化(AR)

[対策]

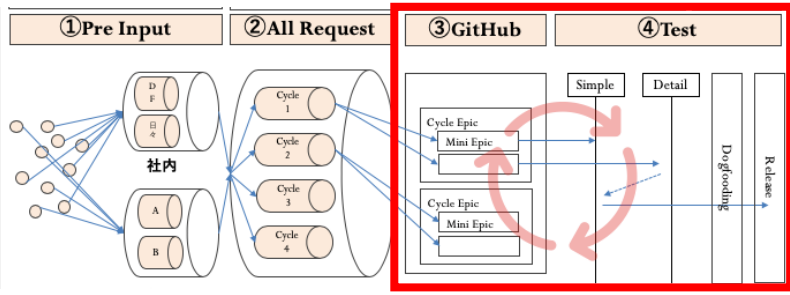
- 時期/背景/方針/対応すると顧客提案がしやすい等で優先付
- ①は粒度が均一でないため、統一した粒度やボリュームに分割
- Issue化直前のフェーズとなるのでチケットに必要な情報は全て載せる

[結果]

- いつ何を対応するか整理されているので、チケット化の時間短縮
- GitHub上だと一覧性が確保できないためSpreadsheetで一覧化や絞り込みが可
- Sprintの長期計画が立てやすく管理工数の削減

リードタイム短縮！

5.フローでの対策と結果



③ Issue内に明確な仕様策定を行いDevと壁打ち(GitHub)

④簡易テスト→詳細テストを実施、FBも③と同じ場所へ

[対策]

- Sprint Planning前にPOとSM(POサブ)で仕様のドラフトを作成
- Sprint PlanningでDevと工数や難易度を確認して、適切なベロシティを策定
- Cycle単位にCycleEpic(親)>PlanEpic（子)>QABug(Issue)とツリー構造化
  - 親EpicにCycle方針を記載
  - 子EpicにSprintPlanningで決定した内容を記載し親Epicに紐付け
  - 子Epicをベースに確認を行い、Bugが見つかればQA_BugとしてIssueを子Epicに紐付け

Git

[結果]

- 事前PlanningによりPlanning本番で最終合意だけとなりスムーズ
- DevチームはGitHub上で作業が完結するため作業工数の短縮
- どのEpicで比較的Bugが多い少ないを判断できるのでテスト濃度を調整も可能

リードタイム短縮！

その１. 継続的により良い解決方法を探している状態

- 今回のフローに落ち着いたのは何度もトライ＆エラーでブラッシュアップしてたどり着いた姿である
- 現場環境やメンバーのレベルアップにより効率的にチームが稼働できる形を常に追い求めている

その2.　ベスト存在しないが、ベターな手法を生み続けることはできる

- 適切な回答を続けるだけでもリードタイムは大幅に削減できる

その3. テストはクリエイティブな仕事

- 仕様策定、開発依頼、テスト設計/実行、ファシリ、コネクション、CS…
- テストに留まらず全工程的に参画し自分がプロジェクトを動かしているマインドセット
- 下流の枝葉に限られた品質保証ではなく、より根本的な品質管理/課題解決が可能になる
- できることは無限にあり、より良い品質保証を追い続けて常に現場を改良していきましょう

大規模なチーム体制で目指すAgile

自己紹介

上野　彩子

(うえの　　あやこ)

品質・技術統轄部
技術推進部
アジャイルファンクション

- SIerにてBtoC向け製品第三者検証、購買システム開発の要求定義からテスト、保守運用まで経験。
- SHIFT入社後、基幹システム刷新プロジェクトに参画。
- QAチームのアジャイルマインドを強化する傍ら、PO支援に入ることで業務設計および継続的リリースに貢献している。

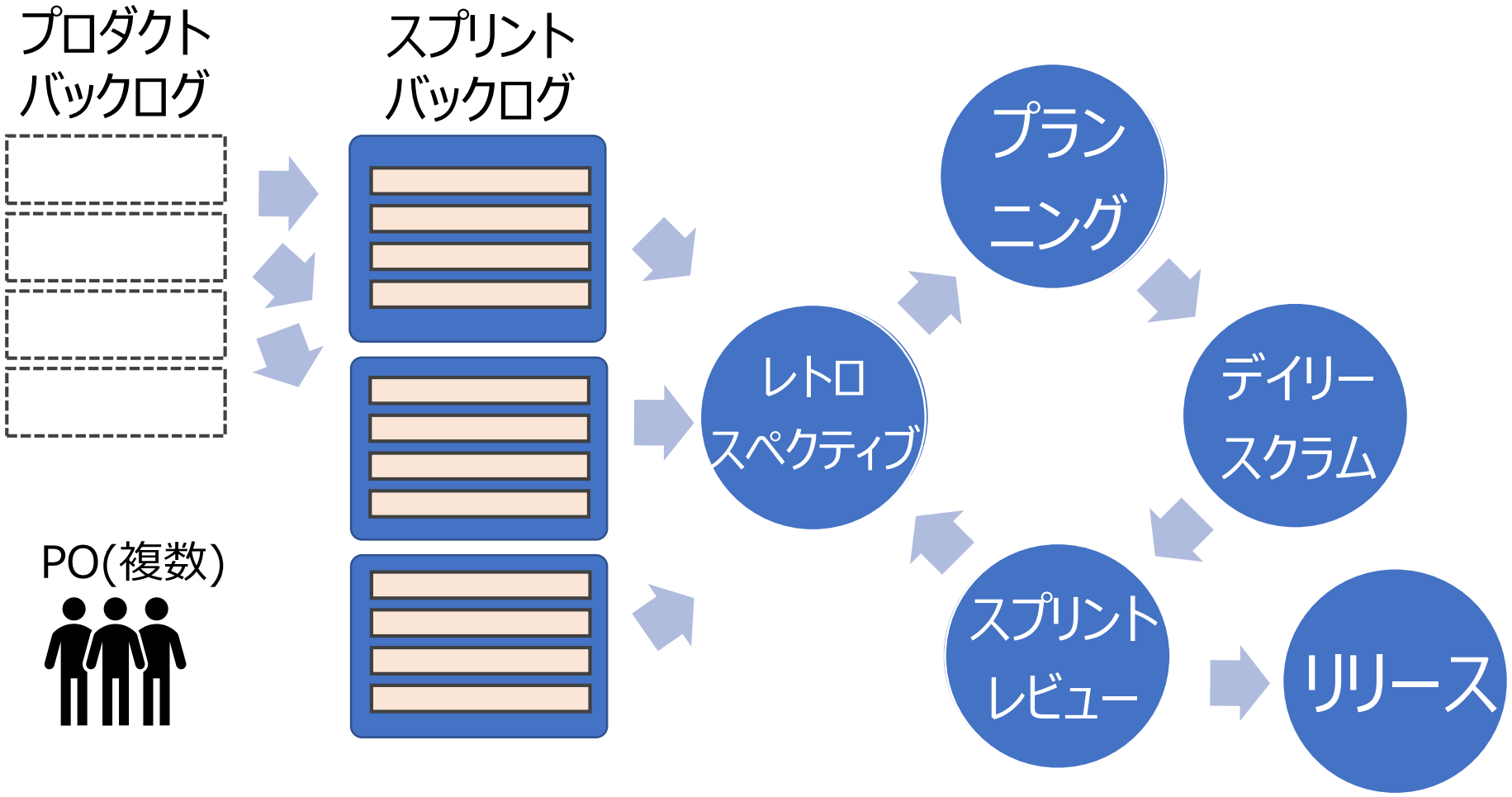
AGENDA

1. はじめに
2. 現場の悩み
3. 解決に向けたアプローチ
4. まとめ

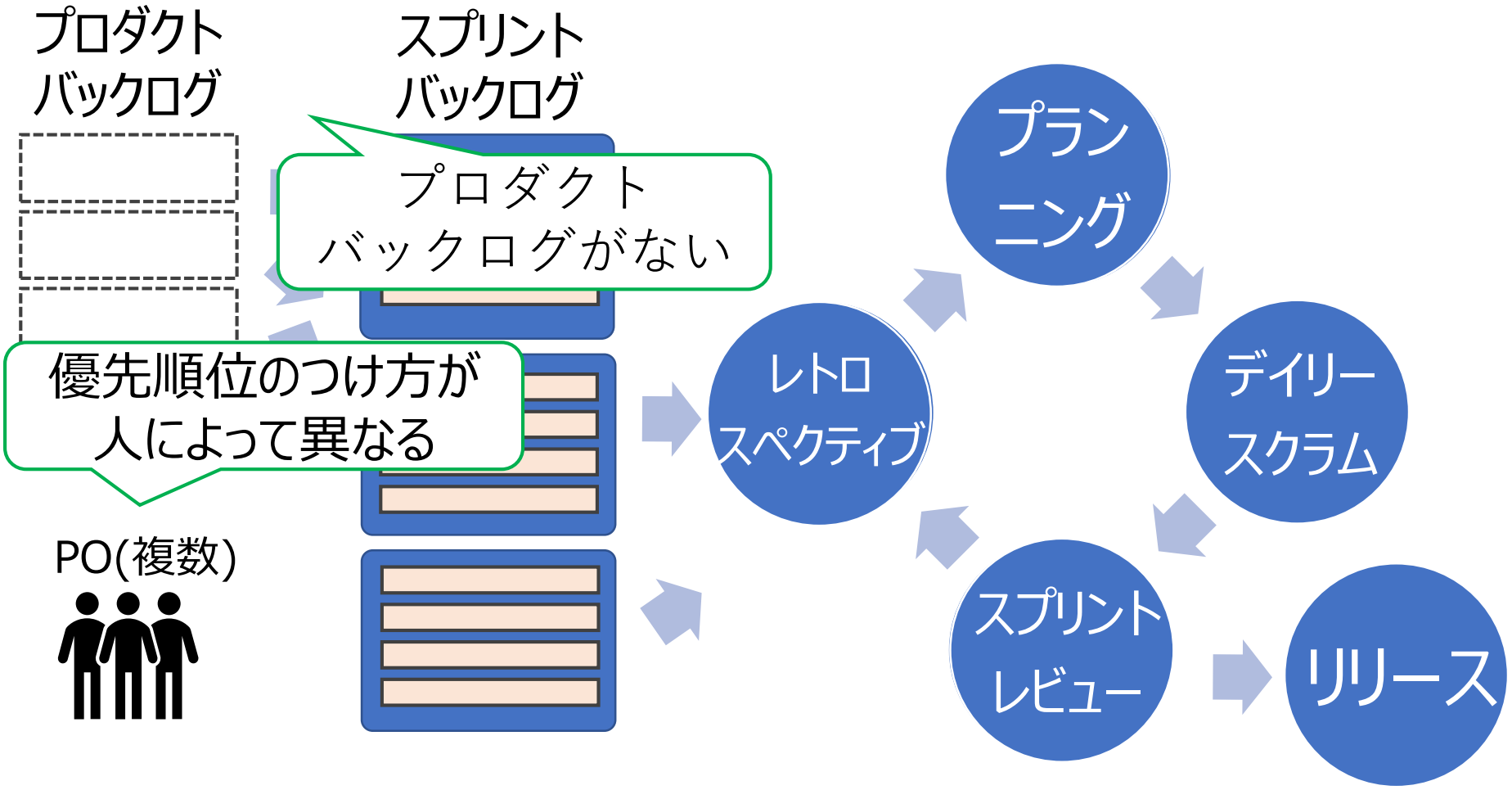
担当プロジェクトの概要

- プロジェクト概要
  - 長期利用をしてきたレガシーシステムのリプレイス案件
  - レガシーシステムと並行稼働しながら、段階的にリプレイスしていく

チーム体制変更により顕在化し始めた課題



チーム体制変更により顕在化し始めた課題

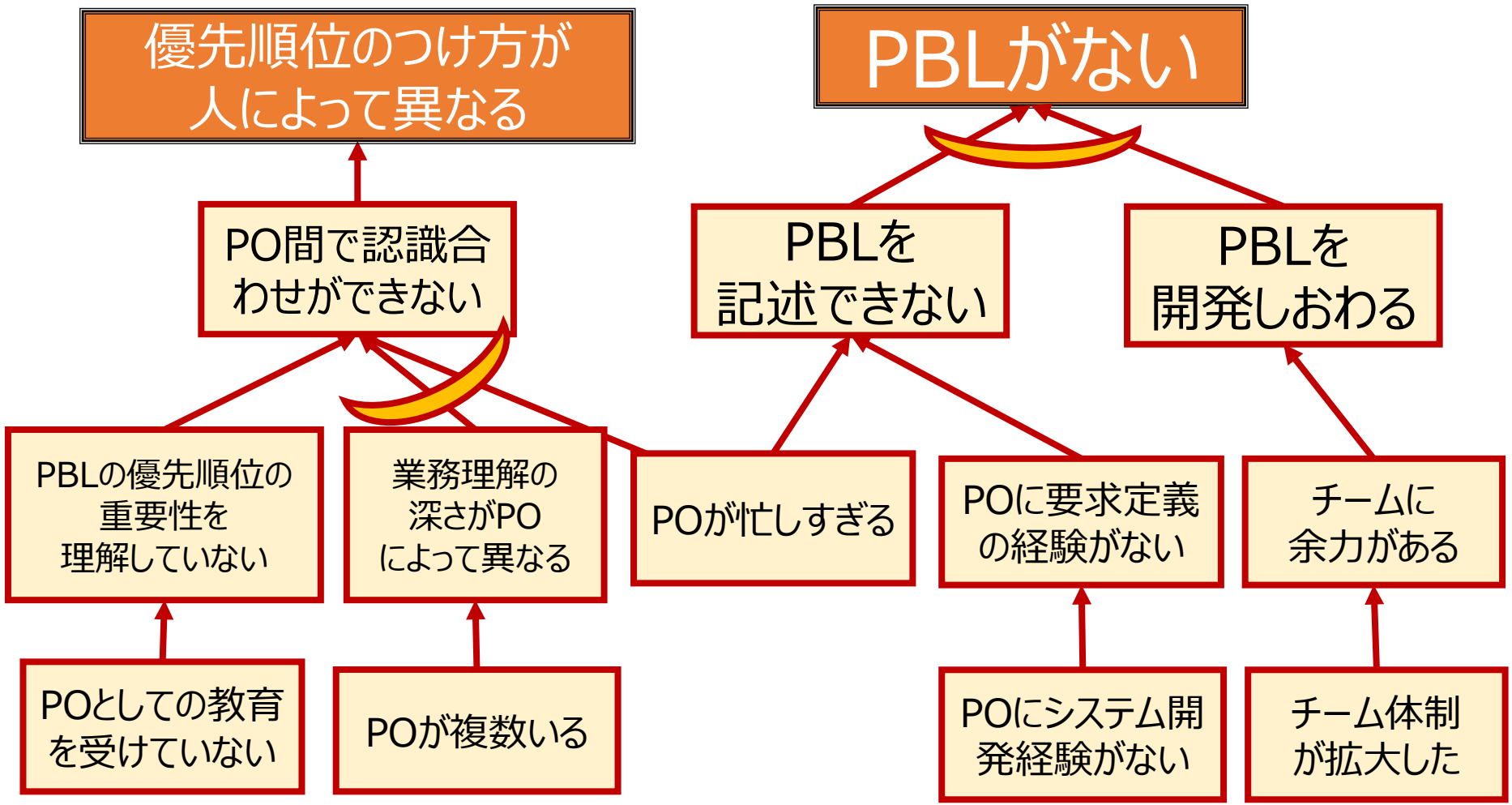


チーム体制変更により顕在化し始めた課題

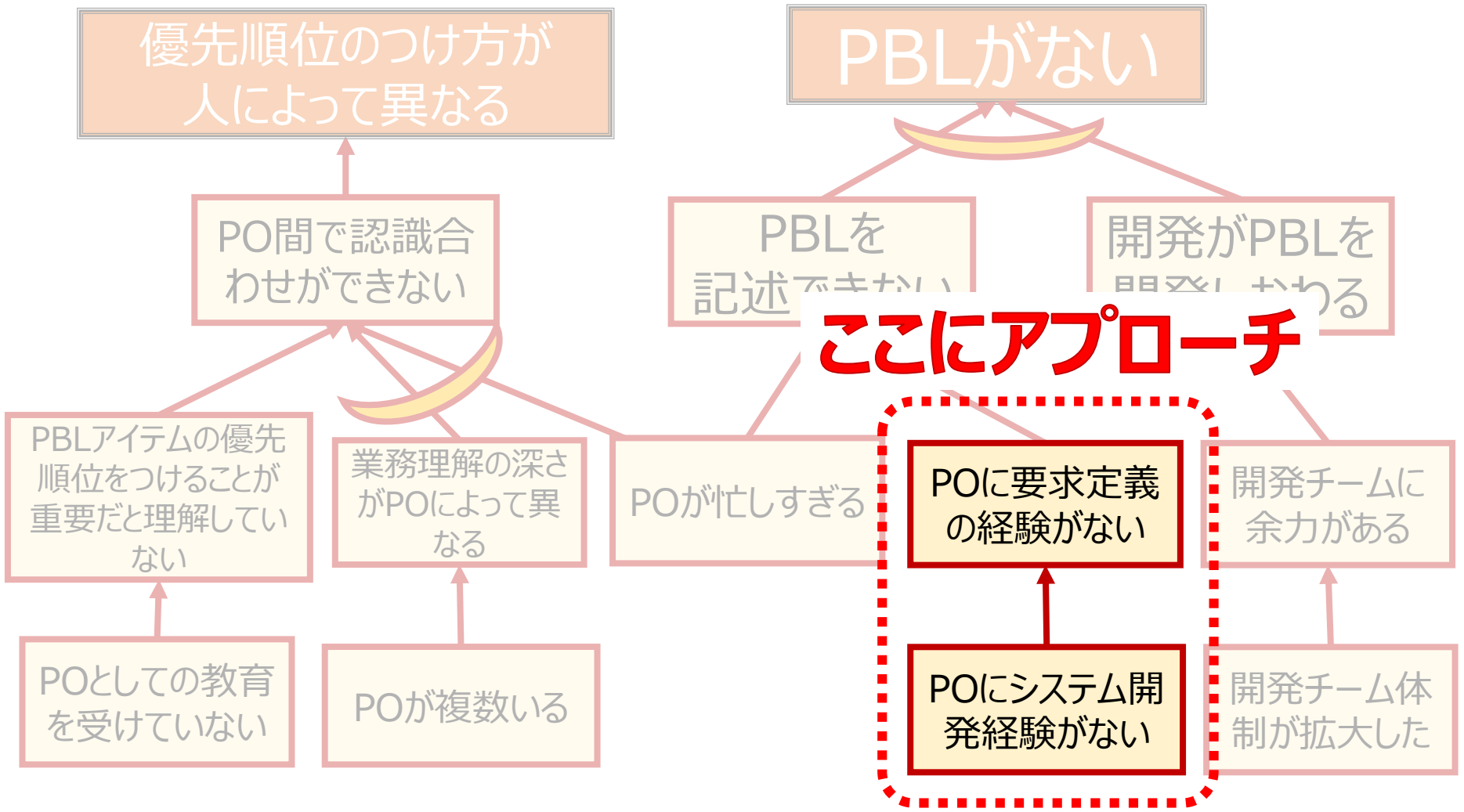
- プロダクトバックログ（PBL）がない
- 優先順位のつけ方が人によって異なる

望ましいタイミングに
リリースができないリスク

課題の原因をロジックブランチを用いて分析



根本原因を回避して課題を解決する



プロダクトバックログがない問題へのアプローチ

プロダクトバックログがないなら、
開発に追いつかれる前に作ればよい

- なぜPOはプロダクトバックログを作ることができないか？
  - POはシステムを作ったことがない
  - 課題を発見し、To-beを見つけるプロセスに慣れていない可能性がある

プロダクトバックログがないと、
望ましいタイミングでのリリースは不可能

プロダクトバックログがない問題へのアプローチ

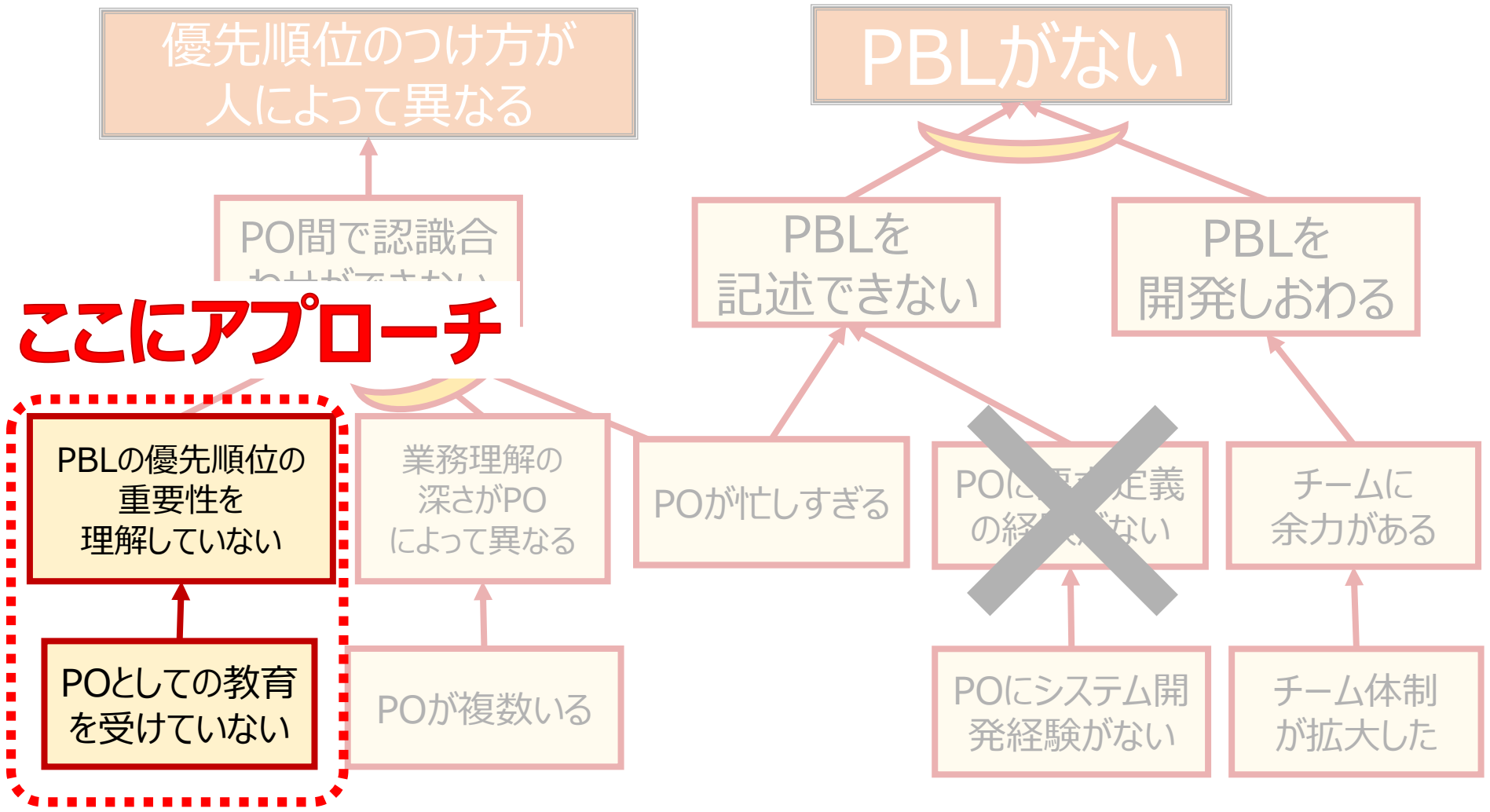
POに要求定義の経験がなくてPBLを出せないなら、
チーム全体でフォローすればよい

- 仕様を決定する場づくり
  - PO、開発チーム合同の会議体の設計
- As-isの整理
  - POが要件として出せなかった隠れた業務を明らかにする
  - 現状の課題を一緒に探し出す

⇩ その結果…

To-beを一緒にプロダクトバックログに
落とし込めるようになってきた

根本原因を回避して課題を解決する



POによって優先順位が異なる問題へのアプローチ

人によって優先順位の付け方が異なるなら、
PO間で認識合わせをすればよい

- なぜ認識合わせをしないのか？
  - 優先順位をつけることがPOの仕事だと認識していなかった
  - POが忙しすぎて、時間が取れない

- その結果、開発チームは次スプリントの予定が立てられず、工数のロスが発生していた

リリースが遅れる可能性があった

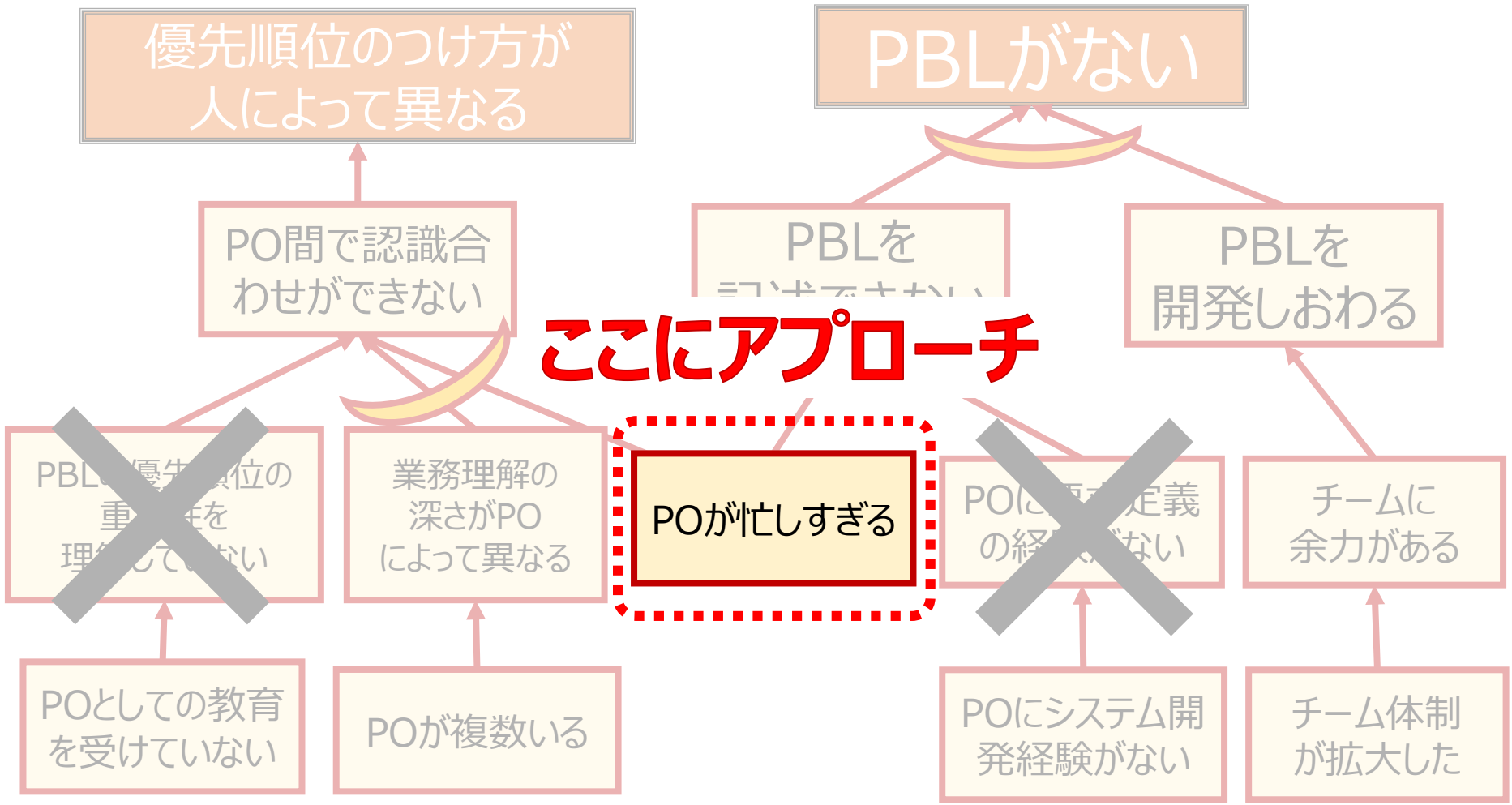
POによって優先順位が異なる問題へのアプローチ

- POと開発者が同席するミーティングで、POに優先順位をつけないことによる開発への悪影響を認識してもらった

⇩ その結果…

優先順位付けをPOの仕事として認識し、
POとしてレベルアップした

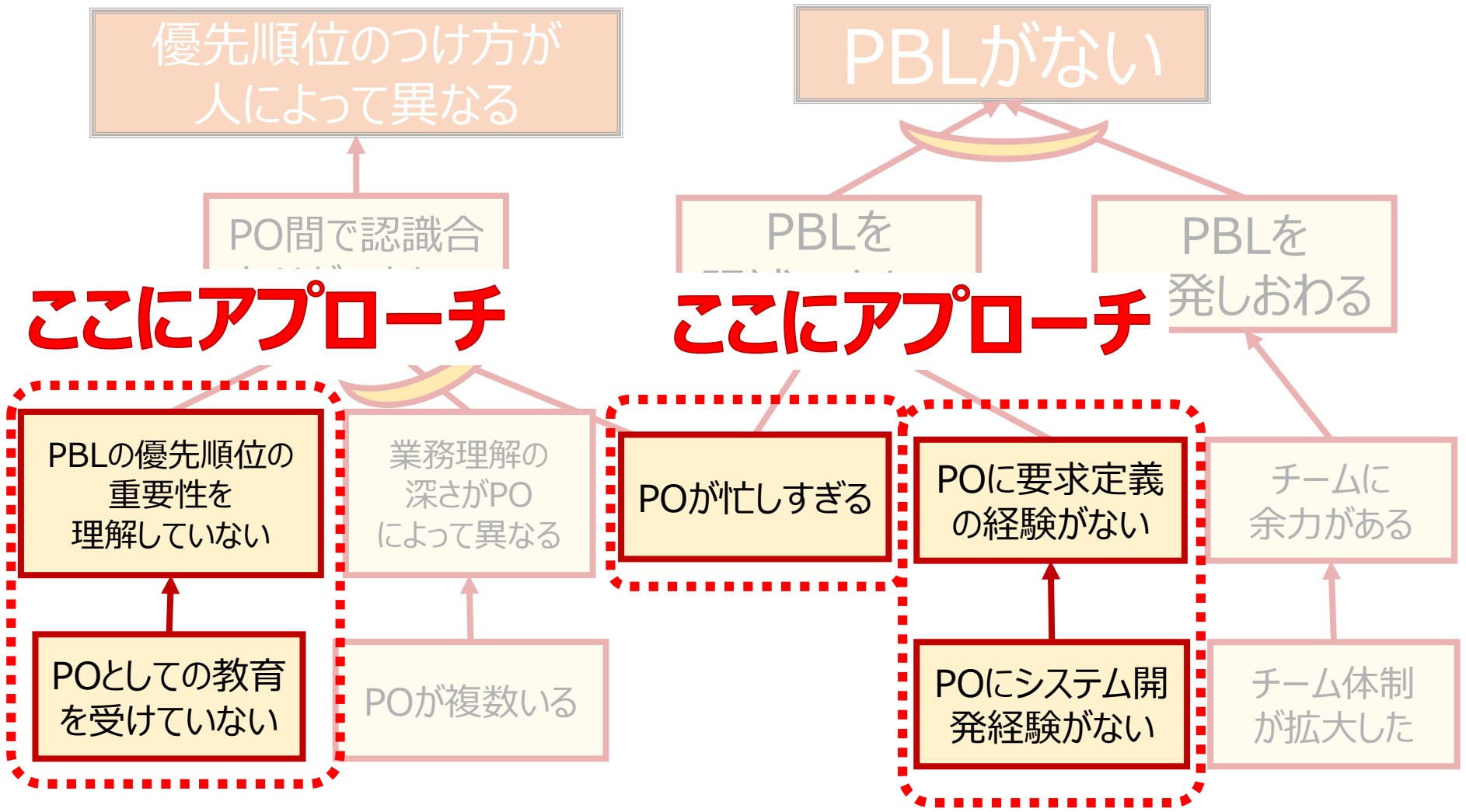
課題の原因をロジックブランチを用いて分析



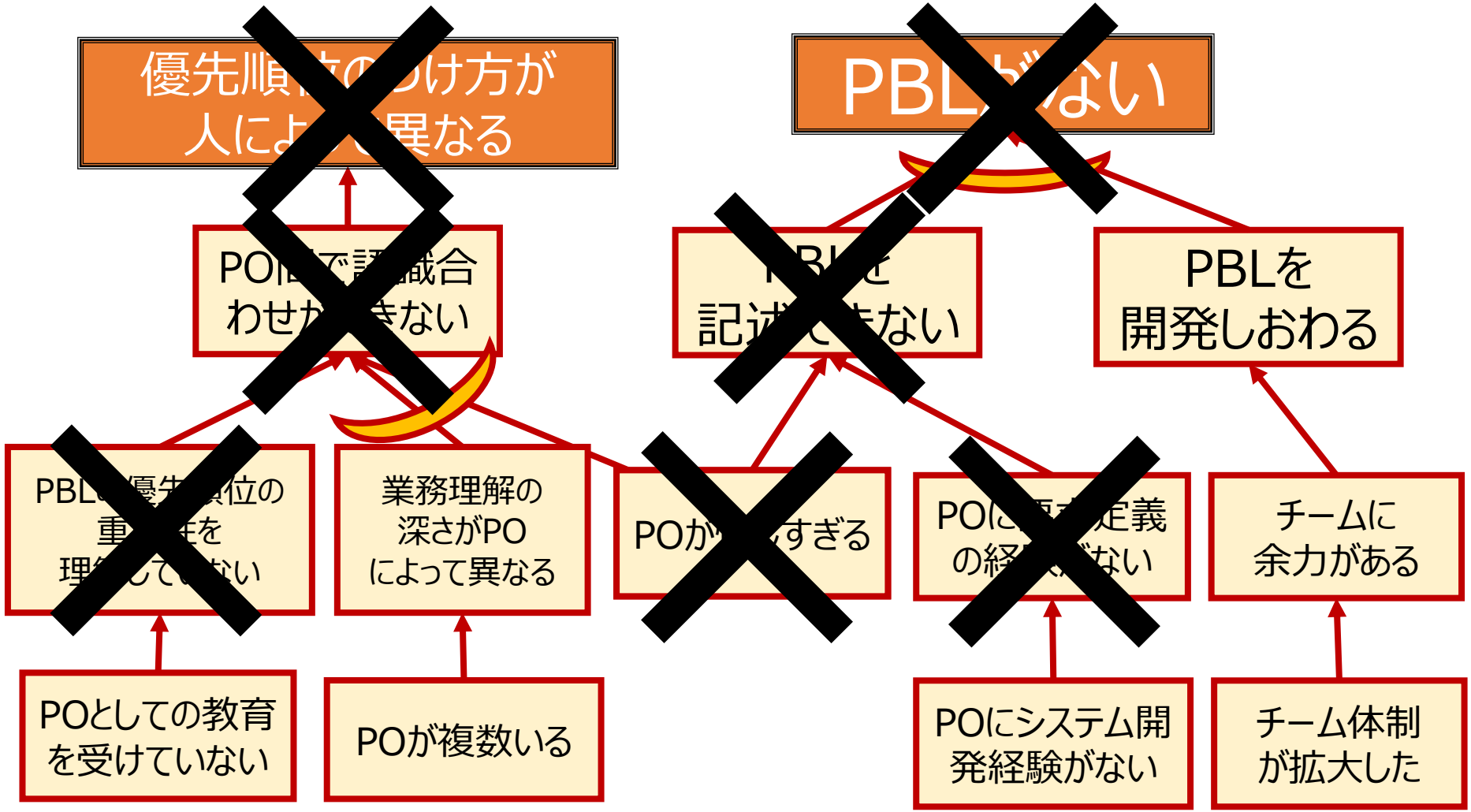
POが忙しすぎる

- 上司へこちらから報告、相談した

根本原因を回避して課題を解決する



根本原因を回避して課題を解決する



アプローチの結果

- POが望むペースでリリースが実現した
  - 業務システムの場合、短いサイクルでのリリースは現場への周知のコストがかかり、望ましくない
  - 業務に支障がない単位に分割することで、現場の負担を最小限にするリリースが実現した

これまでの課題を通じて

- 改善を繰り返した結果、大規模なチーム体制で望ましいペースでのリリースができるようになってきている
- よりよい形を模索しつづけている

アンチパターンと呼ばれる大規模体制でも、
アジャイル開発ができる