DevOpsの価値とプラクティス

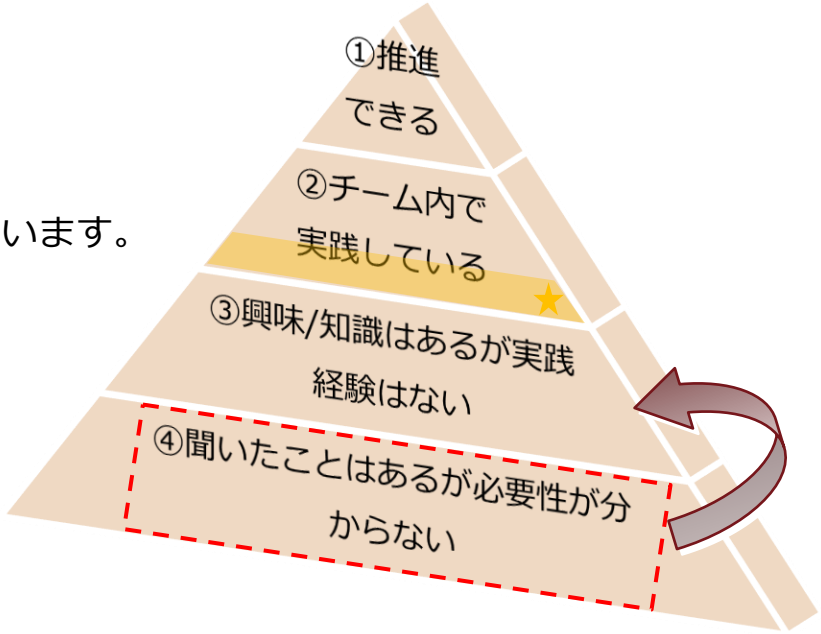
オプスラボ合同会社

はじめに

- 本資料は、当社が以前参画した民間協議会活動のなかで有識者メンバーが議論の積み重ねを整理して作成したものです。作成から時を経ていますが内容は普遍的であり今でも十分活用できるものです。ただしDevOpsに対する”正解”を示すものではないことにも留意が必要です。

- DevOpsを実践することの価値を理解することができます。
  - どのようなITで有効なのか？
  - 実戦して得られるビジネス価値はなにか？
  - 何をすればその価値を得ることができるのか？

- 現時点の本資料は、右図の④向けの方を主な対象読者としています。DevOpsの価値/必要性を理解した④の方が本書を読むことでチーム内で実践できる③の領域への準備ができることを一つの目標としています。また、④以上の領域の方向けにもチーム内等で啓蒙活動をする際に活用いただくことも想定しています。



目次

- 背景: 想定する適用対象のプロダクト / サービス / プロジェクトの特性
- DevOpsの戦略と価値
- 当社が考える “DevOps” のスコープ
- 領域 (担当役割) ごとの価値、阻害要因、効果、プラクティスの関係の詳細
  - 1.アイデアの実現が早くなる
  - 2.必要なものを早く市場に投入できる
  - 3.(Dev)価値を生む作業ができるようになる (ムリ、ムダを減らせる)
  - 4.安全かつ頻繁に自信を持ってリリースできる
  - 5.自己組織化された強いチームを作ることができる
  - 6.(Ops)価値を生む作業ができるようになる (ムリ、ムダを減らせる)
  - 7.次に何をすればよいかわかり改善できる
  - 8.市場のニーズとの乖離がわかる
  - 9.ビジネス目標に沿って協力できる
  - 10.チャレンジできる

背景: 想定する適用対象のプロダクト / システム / プロジェクトの特性

テクノロジーやデータを活用した新たな価値を提供することを目的とし、
顧客やエンドユーザーが直接利用できるサービスを、
新しく開発、提供するプロジェクト

■先が見えない 予見できない

- サービスとして提供されるべき機能や利用シナリオは不明確であり、正解はない

■競争が激しい

- 国境や市場の壁が低く (あるいは無く) 国際競争・企業間競争に晒される
- テクノロジーの特性上、スタートアップに代表される小規模／スピードのある組織の参加が容易で参入障壁が低い
- あらたな市場やエコシステムの構築においてスピードが重視される

■変化が激しい

- よりエンドユーザーに密着しており時代背景や流行により左右される
- 技術の変化が激しくそれにより実現可能となる内容に影響される

この種のプロダクトはSoE （ Systems of Engagement）的と呼称されます

DevOpsの戦略と価値

ビジネスとして成功する確率を高めるため、
利用者へサービスの提供と、そのフィードバックに基づく学びと対応を、
適切なバランスの品質を維持した上でスピードを持って実現できるようにする

- 先が見えない 予見できない
- 競争が激しい
- 変化が激しい

フィードバックに基づく学びと対応

スピードを持った利用者へ提供

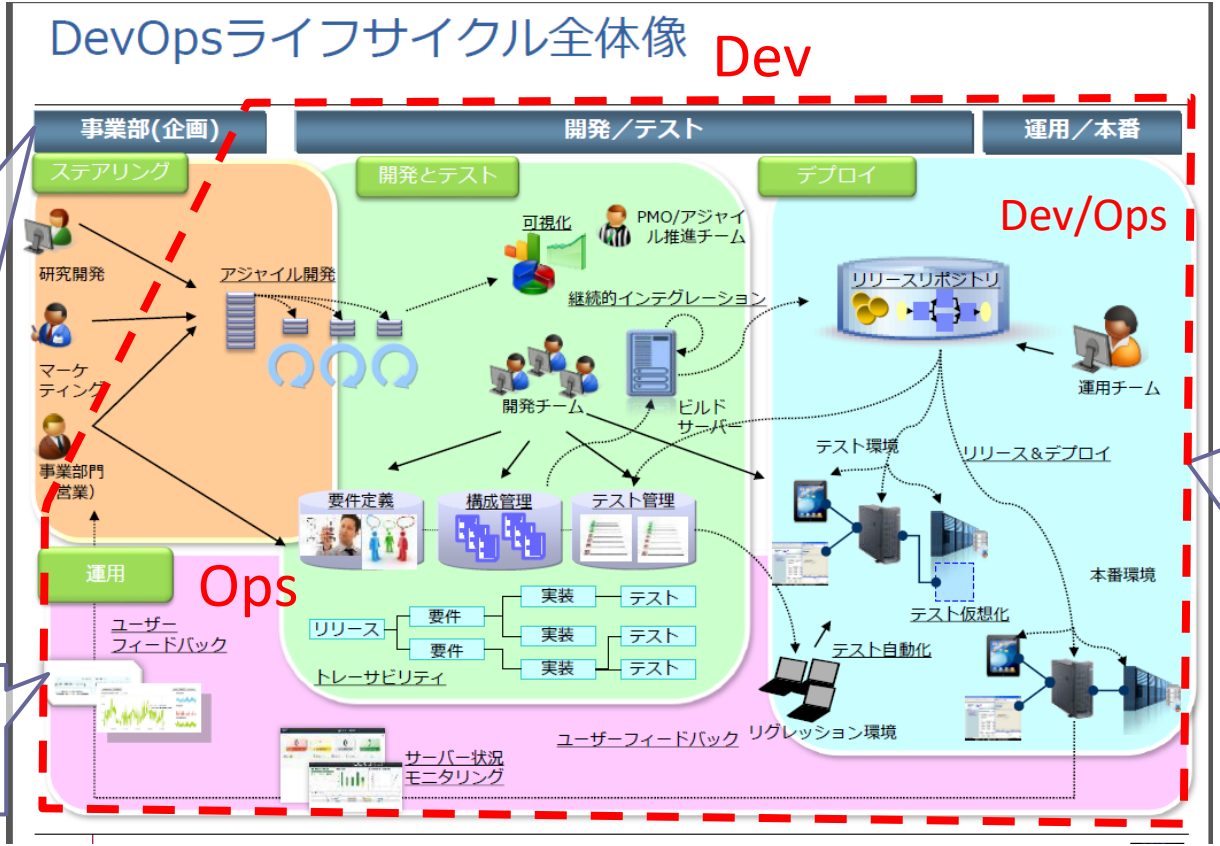
適切なバランスの品質の維持

本資料での “DevOps” ：スコープ

Biz

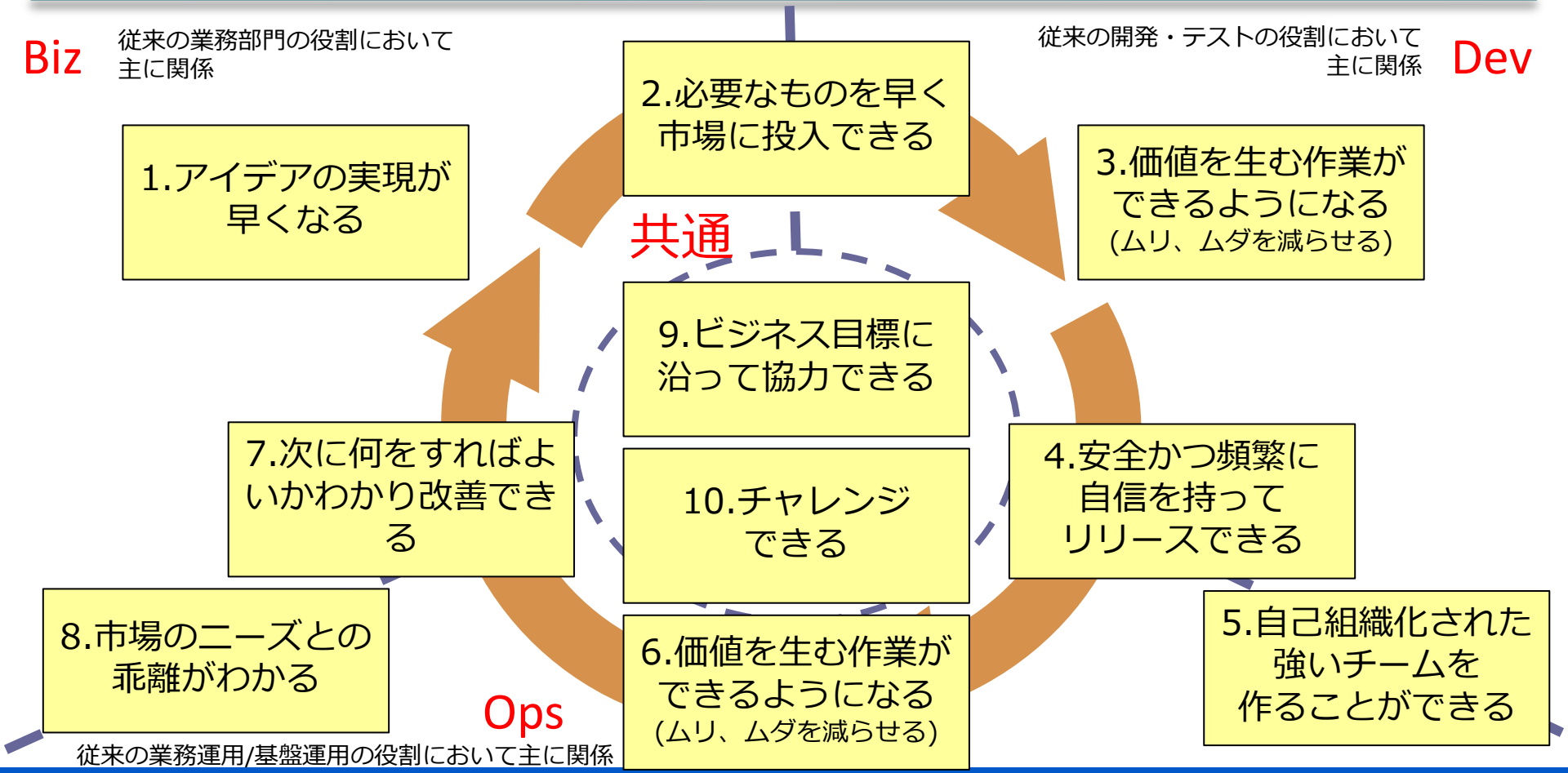
プロジェクトにおける
ユーザー企業
(開発したシステム、
の利用 / あるいはそれを
使用したサービスの
提供主体) の
業務/企画担当
の関わりは含むが、
特定のプロダクト、
プロジェクトの範囲まで
とする

フィードバックを効果的
に収集し計画に反映
する方法など



運用には
基盤観点でのIT運用と
、業務観点でのサービス
運用 (ユーザー・サポート
など) の両方が含まれる

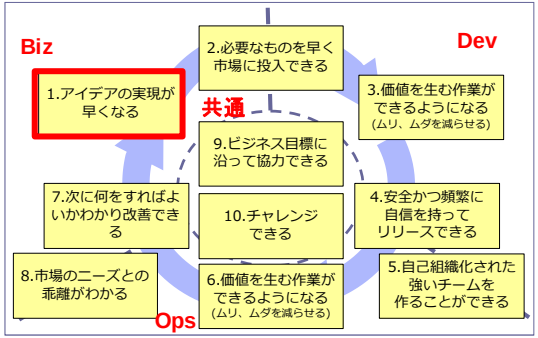
DevOpsの価値とマッピング



1. アイデアの実現が早くなる

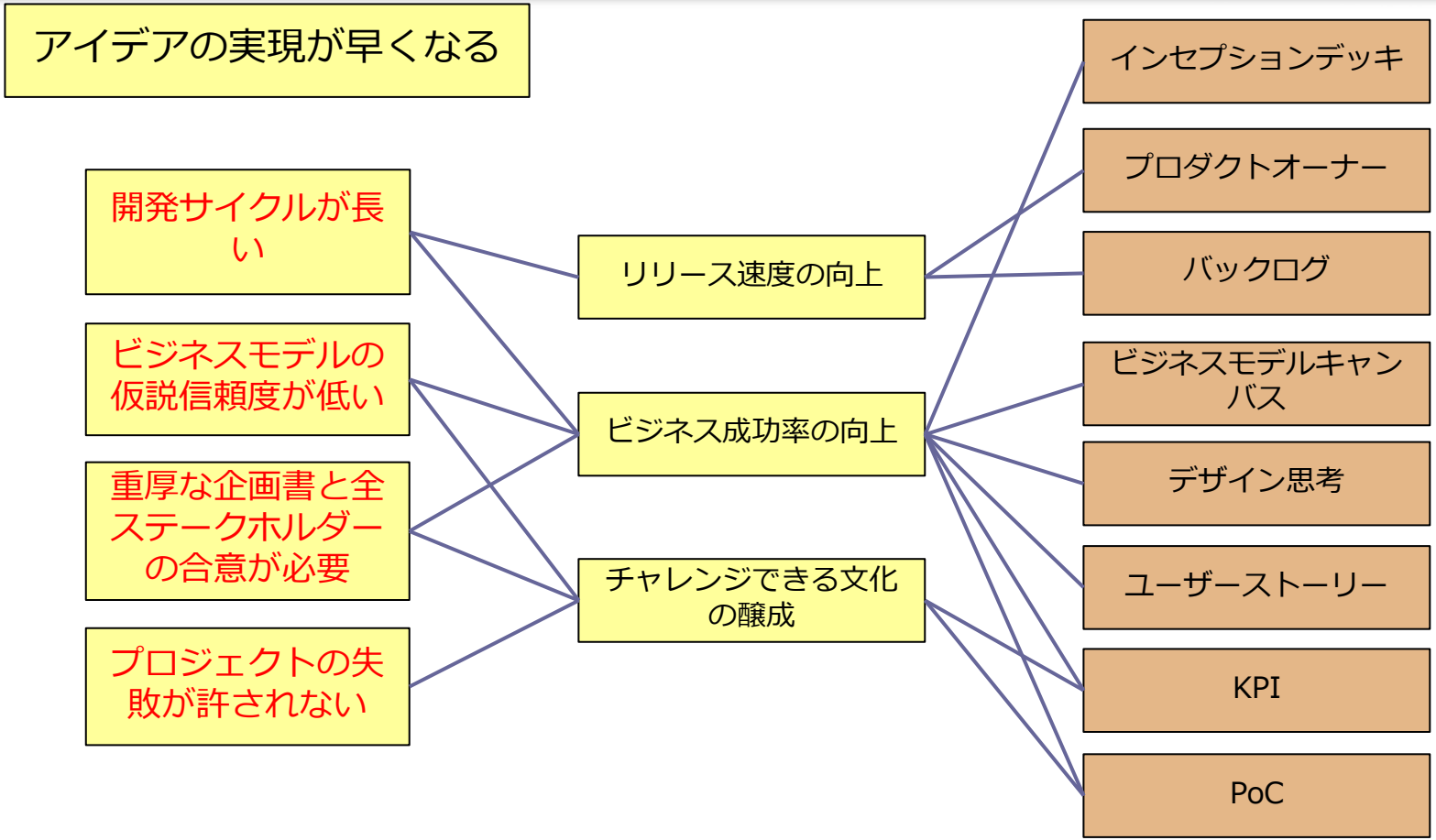
■ 価値
- サービスを市場にリリースして効果実測が行える。
  またフィードバックが得られ新たなニーズを発見できるため機会損失が小さくなる

■ 課題従来手法における阻害要因(失敗例)
- 開発のサイクルが長い、必要な時に開発者が空いていない
- ビジネスモデルの収益計画(仮説)の信頼度が低い
- プロジェクトを始めるには重厚な企画書と、全ステークホルダーの合意が必要
- プロジェクトの失敗が許されない



■ プラクティス
- **「ビジネスモデルキャンバス」**はビジネスの構造を9個の領域に分解し、事業の課題や方向性の理解を助け、開発を進めやすくする効果が得られる。
- **「バックログ」**を明確化することで作業の優先度が明確になる効果が得られる。
- **「デザイン思考」**はユーザの価値に注力し、問題の本質を明らかにし、ユーザが本当に求めるものを明確にできる効果が得られる。
- **「インセプションデッキ」**を使うことで、プロダクトの目的・ビジョン・方向性の理解を共有化できる効果が得られる。
- **「PoC」**とは実現可能性の検証である。PoCを行うことでスポンサーはリスクを軽減させることができ、事業者は資金提供者へのアピールをすることができる。

価値、阻害要因、課題、プラクティス



プラクティス (抜粋)

- ビジネスモデルキャンバス
  - 事業の構造を理解するために9個の要素に分解する。その結果、事業の課題や方向性を理解して開発が進めやすくなる。
  - 9個の要素：1. 顧客、2. 与える価値、3. チャネル、4. 顧客との関係、5. 収益、6. リソース、7. 活動、8. パートナー、9. コスト
- デザイン思考
  - ユーザーの価値に注力し、問題の本質を明らかにするために5個のステップを実行する。その結果、ユーザーが本当に必要とするものを生み出すことができる。
  - 5個のステップ：1. 共感、2. 問題定義、3. 創造、4. プロトタイプ、5. テスト
- インセプションデッキ
  - プロダクトの目的や方向性が曖昧なまま開発を進めている場合には、プロダクトの目的や方向性を明らかにした10の質問に回答しチームで共有する。その結果、チームはプロダクトの目的、ビジョン、方向性を理解して開発が進めやすくなる。
  - 10の質問：1. われわれはなぜここにいるのか、2. エレベータピッチを作る、3. パッケージデザインを作る、4. やらないことリストを作る、5. 「ご近所さん」を探せ、6. 解決案を描く、7. 夜も眠れなくなるような問題は何だろう？、8. 期間を見極める、9. 何をあきらめるのかをはっきりさせる、10. 何がどれだけ必要なのか

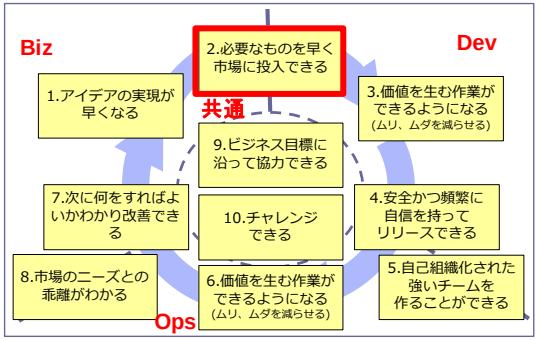
参考：IPA アジャイル型開発における プラクティス活用リファレンスガイド

2. 必要なものを早く市場に投入できる

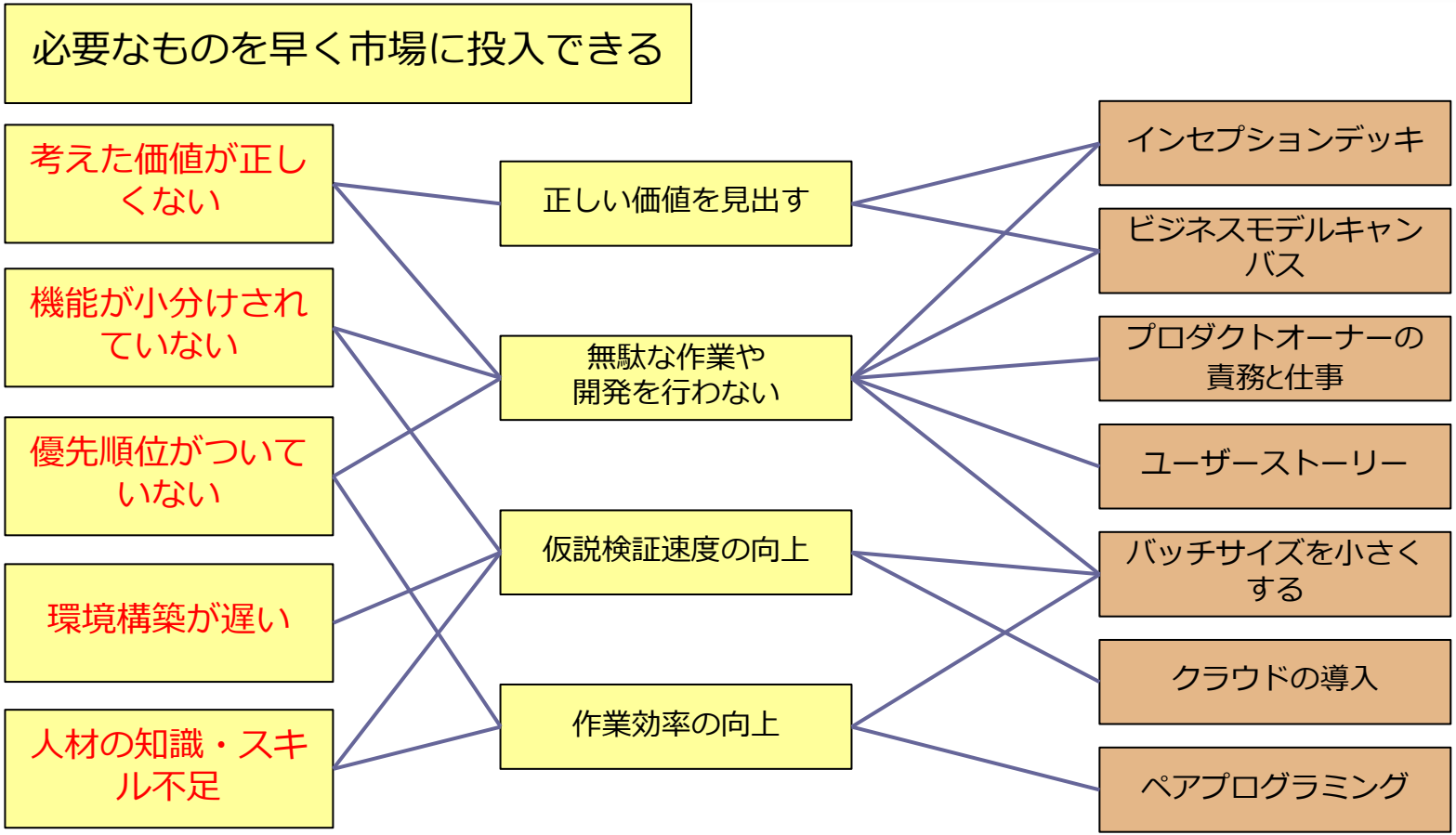
■ 要約
- 必要なものを見極め、優先順位を正しくつけて作業を進めることにより、必要なものを必要な分だけ必要なものから市場に投入することができる。スピーディーに作業を進めつつ効率的な作業を行うことにもつながる。

■ 従来手法における阻害要因(失敗例)
- 全体像の把握が出来ておらず、考えた価値が正しくなかった
- 開発機能の優先順位が正しく設定手出来ていなかった
- 機能が小分けされていなくスピード向上につながらなかった
- 環境構築ができておらずスムーズな開発が出来なかった
- 人材の知識やスキルが低く効率的に進まなかった

■ プラクティス
- **「インセプションデッキ」**や**「ビジネスモデルキャンバス」**を取り入れ、プロジェクトの目的や背景等の全体像を理解し、正しい価値を共有する
- **「ユーザーストーリー」**を用いたり**「バッチサイズを小さくする」**ことで必要な機能を適切に小分けする
- **「クラウド」**を導入し素早い環境構築を実現する



価値、阻害要因、課題、プラクティス



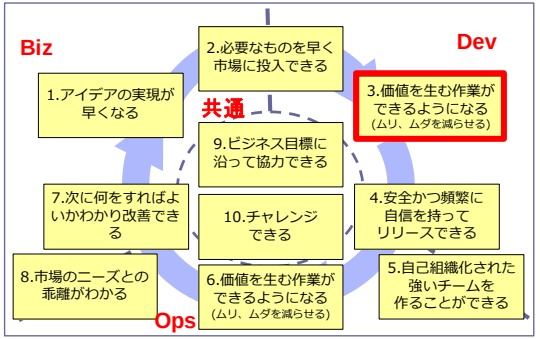
プラクティス (抜粋)

- プロダクトオーナー
  - 要件の優先順位や仕様がなかなか決まらない場合は、プロダクトの結果に責任を持ち優先順位や仕様を決める役割を設ける。その結果、プロダクトについての決定を素早く行うことができる。

- ユーザーストーリー
  - プロダクトの全体像が見えない、何が重要なのかがわからない場合は、ユーザーストーリーを利用者の時間軸と、優先順位の2つの軸でマッピングしよう。その結果プロダクトの全体を俯瞰でき、計画づくりに役立つ。

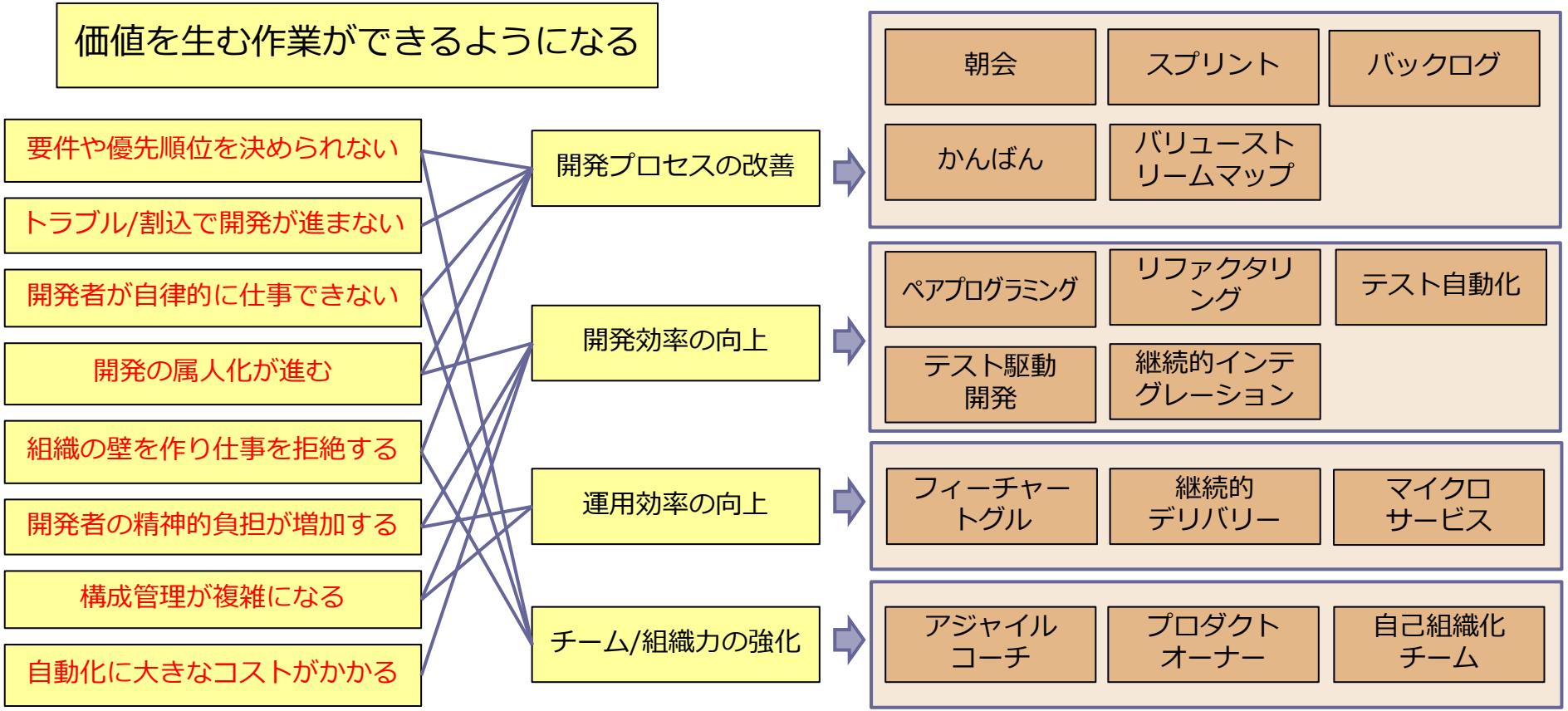
参考：IPA アジャイル型開発における プラクティス活用リファレンスガイド

3. 価値を生む作業ができるようになる



- 価値
  - 無駄をなくして機能と品質の作り込みに集中できる
- 課題従来手法における阻害要因(失敗例)
  - 事業部門が要件や優先順位を決められない
  - トラブルや割込み作業が発生し、予定していた開発作業が進まない
  - 開発者が自律的な開発の進め方に対応できない
  - 自動化を構築・継続していくために大きなコストがかかる
  - 開発者の作業領域が広がり属人化が進む
  - 組織の壁を作り、領域外（境界線）の仕事を拒絶する
  - 機能改修、機能追加を短期で繰り返すため構成管理が複雑になっていく
  - リリース頻度が増えるため開発者の精神的負担が増加する
- プラクティス
  - **「スプリント」**は開発を実施する単位で、スプリントを繰り返しながら開発を進めていく。一つのスプリントは数週間の短い期間で設定し、その中で設計からテスト、レビューまでを実施するため成果が早期に得られる。
  - **「テスト自動化」**は同じテストを繰り返す時間の削減、人間が起こすミスを防止する効果がある。短期的には自動化を行うための作業量が増えるため長期的な開発で効果を発揮する。
  - **「マイクロサービス」**は機能別に小さなサービスを開発してそれぞれを連携させてシステムを構成する。サービス間は疎結合のためメンテナンス性に優れ、部分的な切り離しなども行いやすい。
  - **「自己組織化チーム」**はメンバーが指示を受けてではなく自律的に仕事を行い、最大限の成果を出すチームである。このようなチームを作るには**「アジャイルコーチ」**による支援が有効である。

価値、阻害要因、課題、プラクティス



プラクティス (抜粋)

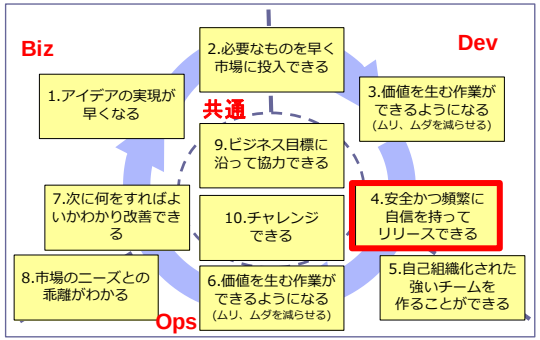
- かんばん
  - プロジェクトを取り巻く状況が不安定で計画を頻繁に変更せざるを得ない場合には、同時に開発する機能の数を制限して流量コ

ントロールする。その結果、機能を流れるように提供することができ変化に対応しやすい。

- ペアプログラミング
  - チームの中で知識やコードの共有ができていない場合は、ペアを組んで作業をする。その結果、開発チームの中で業務知識やコードについての知識が共有でき、品質や作業効率も向上できる。
- フィーチャートグル
  - コードの中でフラグを利用して機能のオン/オフが切り替えられるようにしておく。その結果、リリースしない機能もコードに含む運用が可能になりブランチを削減できて、構成管理の複雑化が回避できる。
- アジャイルコーチ
  - もしチームが初めてアジャイル型開発に挑戦しようとしているなら、アジャイルコーチに導入を手伝ってもらうべきである。その結果、より確実にチームがアジャイルへの変化を体験できる。

参考：IPA アジャイル型開発における プラクティス活用リファレンスガイド

4. 安全かつ頻繁に自信をもってリリースができる

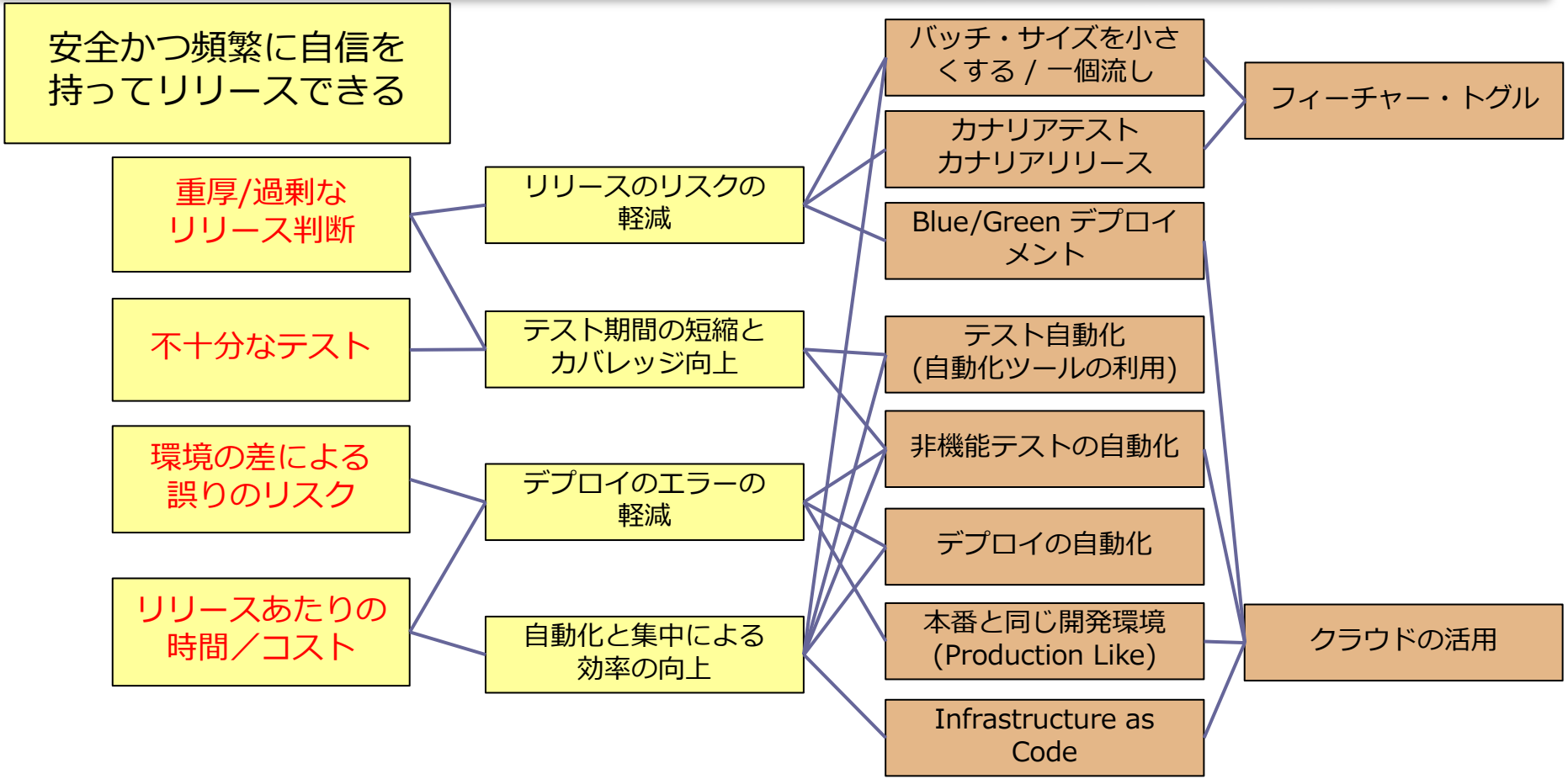


■ 価値
- (Biz-Dev-Ops で構成される) チームは、新しい機能や修正を、高い頻度でエンドユーザーが利用可能な本番環境へとリリースすることができる。チームはそのリリースは、リリースによる負の影響が少なく安全であることに自信を持てる。

■ 従来手法における阻害要因(失敗例)
- リリース判断の重厚化 / 長期化による頻度の低下
- 不十分なテストによる悪影響の可能性
- 環境の差による誤りのリスク
- リリースあたりの時間／コストが高く頻度を上げることが困難

■ プラクティス
- **「バッチサイズを小さくする」**ことでリリースあたりの変更量を減らし、**「カナリアテスト」**や**「Blue/Greenデプロイメント」**によりリリース直後の問題の影響範囲の減らすことでリリースのリスクを下げる。
- **「テスト自動化」**の推進により短期間での十分な回帰テストの実施を実現する。
- **「Infrastructure as Code」**により構成された環境や**「本番と同じ (Production like) 開発環境」**に対して開発を行ことで環境の差異の誤りを減らす。**「デプロイの自動化」**やその継続的な実行により環境構成に由来するデプロイの誤りを早期に検出し修正する。
- 一連の自動化により時間と作業量を減らし、**「継続的デリバリー」**(ビルド、デプロイ、テストの継続実行) により問題の早期発見と修正の効率化を行う。**「一個流し」**の考えを取り入れた集中化によりさらなる短期化を実現する。

価値、阻害要因、課題、プラクティス



プラクティス (抜粋)

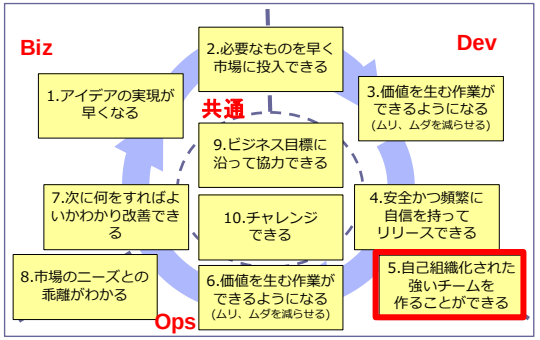
- バッチ・サイズを小さくする / 一個流し
  - 1回のリリースで行う変更量を小さくする (その代わり回数を増やす)。 リリースに含まれる変更の量が少ないほどリリースのリスク (問題が起きた時のインパクト) が低くなる。
  - 多数の変更を同時並行に扱うのではなく、Biz/Dev/Ops のメンバーが短期間で集中して個別の変更に注力して作業を行うことで、切り替えの負担(ムダ)を減らし、誤りの発見と修正を効率よく行うことが可能となる。
  - リリースのバッチ・サイズを小さくするためには、一度のリリース にかかる手間やコストを(リリースの規模によらずにかかる共通のコスト) を減らす必要がある。
- カナリアテスト / カナリアリリース
  - 新しい機能などを本番環境にリリースする際に、全ユーザーにその機能を解放するのではなく、一部のユーザーにのみ機能を解放する。一定期間、状況をモニタリングし、障害が無いことを確認してから他のユーザーへと機能を解放する。これによりリリースのリスクを軽減する。
  - カナリアテストを実施するためには一部のユーザーにのみ新しい機能を解放するような仕組みが必要となる。フィーチャー・トグルはそのための仕組みとして利用できる。
- テスト自動化
  - リリース済みの機能についてのリグレッションテストの実施において、単体、統合、API、受け入れ、のさまざまなレベルでの自動化を行い、テストカバレッジとテストの実行速度を向上させる。継続的インテグレーションにおける単体テストの実行と、継続的デリバリーでのテスト環境へのデプロイ、統合、受け入れテストの自動的な実行と継続的な修正を通じて、リリースにおけるデグレードの不具合を早期に検出して、修正することを可能とする。

5. 自己組織化された強いチームを作ることができる

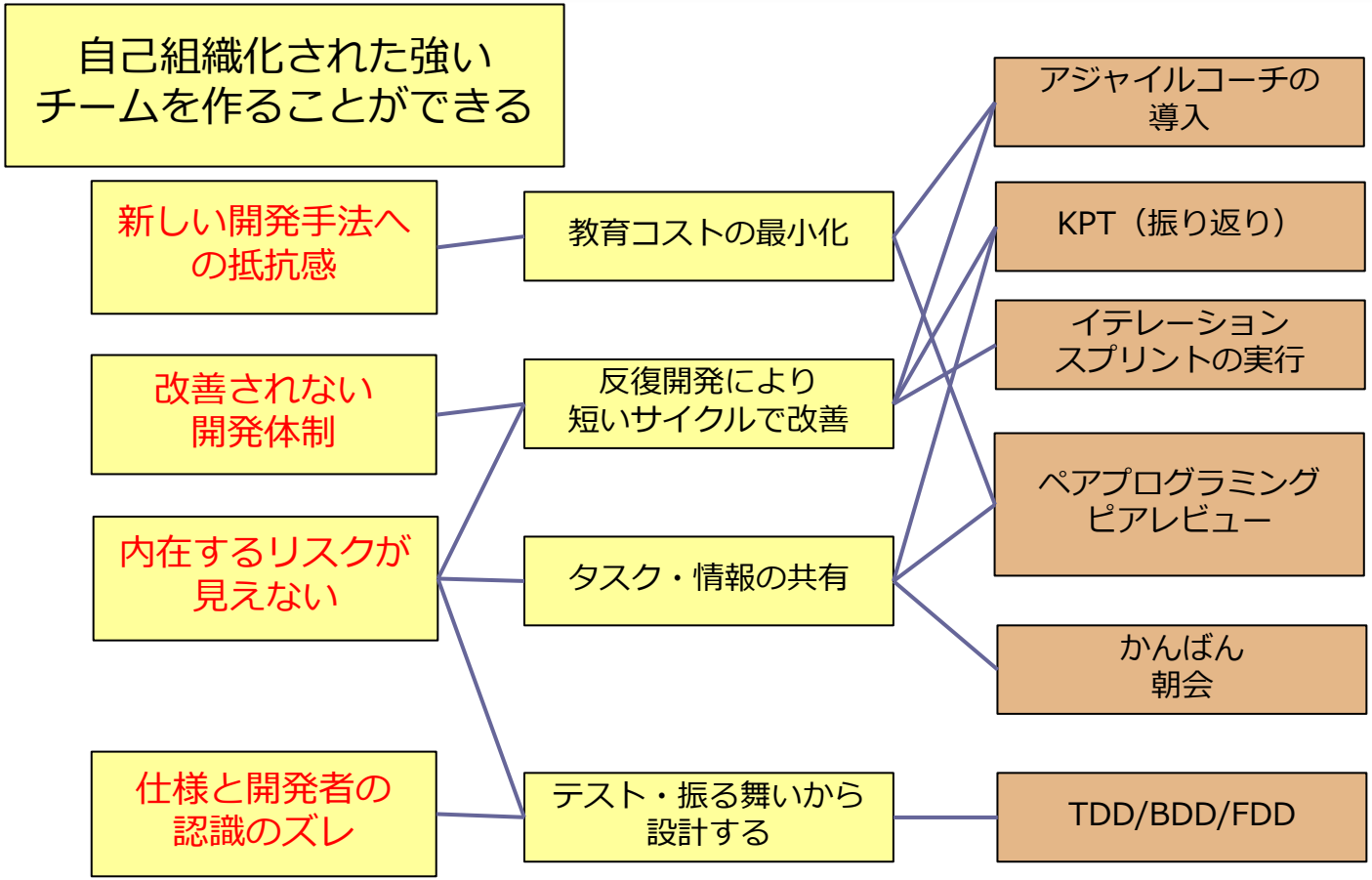
■ 要約
- DevOps チームは、自分たちによって開発手法の選択・開発プロセスの改善を行うことができ、プロダクトを改善するための高頻度なリリースにも対応できるようになる

■ 従来手法における阻害要因(失敗例)
- アジャイル開発のやりかたそのものや、文化の変化が難しい
- 新ツール利用には教育コストがかかる
- 問題点や課題が表面化するのに時間がかかるため改善が遅れる
- タスクや情報が共有されておらず、内在するリスクがわからない
- 仕様と実装の担当者の役割が分断されており、ムダなつくりこみが発生している

■ プラクティス
- **「アジャイルコーチ」**を導入し、新しい開発手法を実践する
- **「かんばん」**や**「朝会」**を通じて情報の共有を行い、**「KPT」**を用いて自ら改善する意識をもたせる
- **「イテレーション」**や**「スプリント」**といった短期間の開発サイクルを回すことで内在するリスクを発見・改善しやすくなる
- **「ペアプログラミング」**、**「ピアレビュー」**を実施し、属人化を防ぎ、教育コストを最小化する
- **「TDD/BDD/FDD」**により、仕様の設計者はテストを、開発・テスト担当者は仕様を意識できるようになる

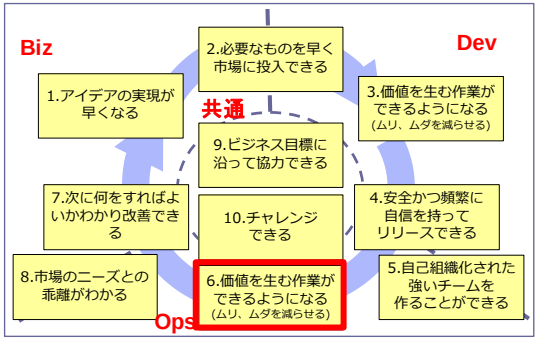


価値、阻害要因、課題、プラクティス

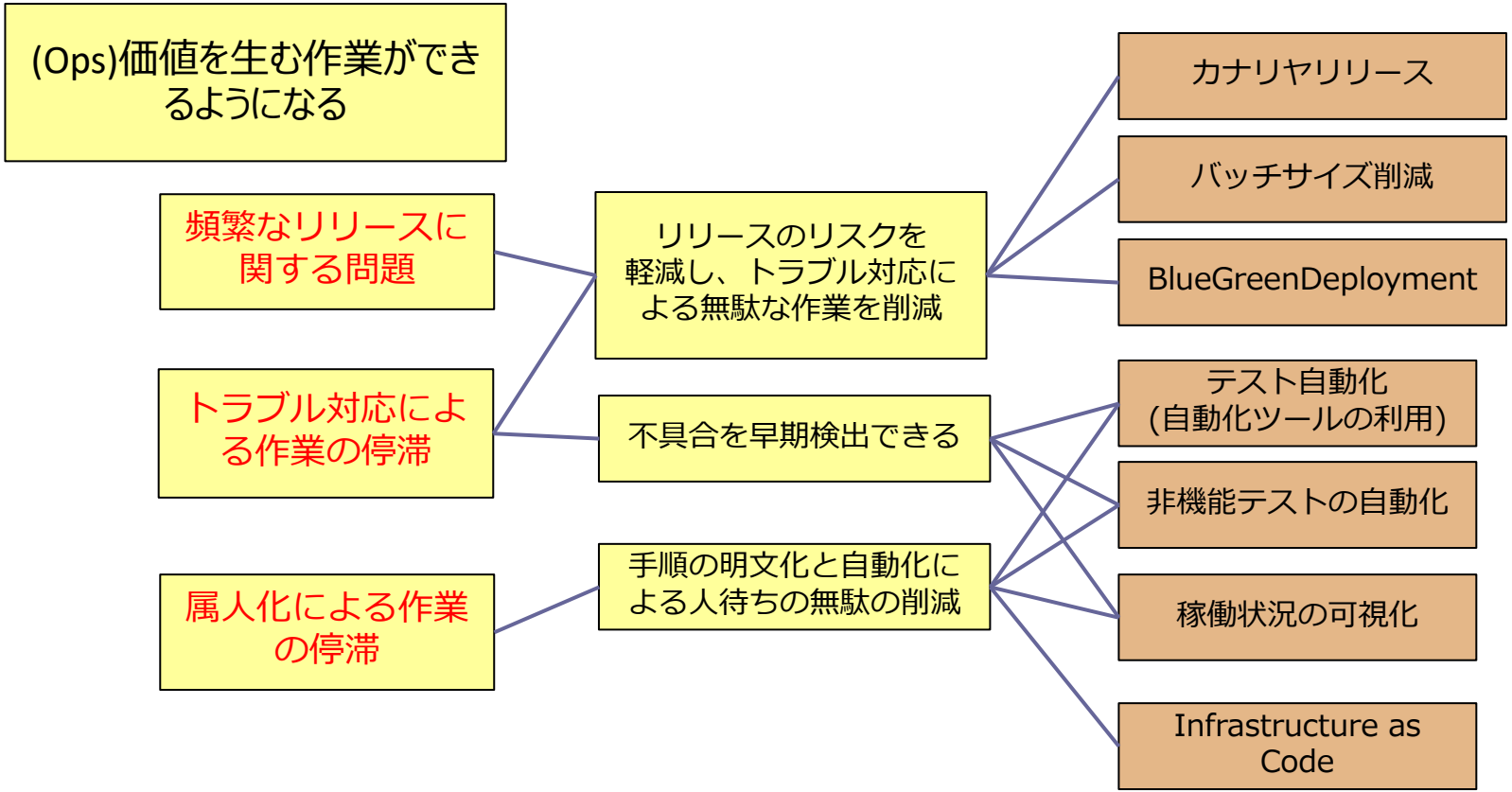


6. (Ops)価値を生む作業ができるようになる

- 要約
  - DevOpsチームは日々の作業における無理、無駄をなくして高頻度なリリースと安定した運用を両立できる
- 課題従来手法における阻害要因(失敗例)
  - システム変更を短期で繰り返すため作業や管理が追いつかない
  - トラブルや割込み作業が多く発生し価値を生み出す作業ができない
  - 運用の属人化が進み作業が滞る
- プラクティス
  - **「バッチサイズを小さくする」**ことで一度にリリースする量を少なくして問題発生時の切り分けや解析を容易にすることができる。
  - **「カナリヤリリース」**はフィードバック獲得とリリースリスク軽減を両立させるため、一部のユーザにだけ新機能を公開する手法である。
  - **「Blue/Greenデプロイメント」**は本番環境と同等の環境をもう1式準備し、リリース時に環境ごとロードバランサで切り替えるリリース手法である。手順が簡素化され復旧作業も容易となる。
  - **「Infrastructure as Code」**はインフラの設定や構築手順をコード化することで作業のミスや属人化を防ぐ手法である。

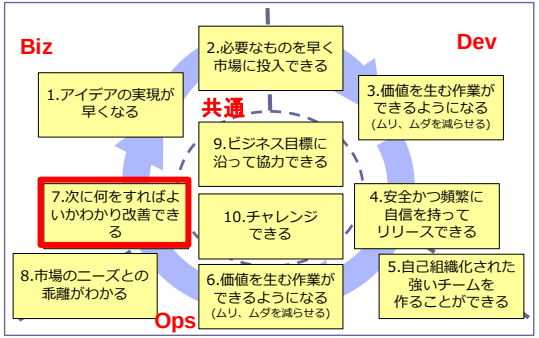


価値、阻害要因、課題、プラクティス



7. 次に何をすればよいかわかり改善できる

- 価値: 市場からのフィードバックにより、作ろうとしているものの価値を正しく理解することができれば、開発の優先順位を適切に決めることができ、ビジネスの成功に結びつかない無駄な開発をすることが無くなる。結果としてチームのベクトルが揃い、ビジョン(価値)を共有することができる。



- 従来手法における阻害要因(失敗例)
  - フィードバックを得ても活用する方法がわからなかった。
  - 開発する機能の優先度を正しく設定できなかった。
  - 開発効率が上がらないボトルネックを特定できなかった。

- プラクティス
  - **「プロダクト・オーナーの責務と仕事」**として、開発が滞らないようにするために、開発すべき機能とその内容について適宜決断を下す。
  - **「バリュー・ストリーム・マップ」**を活用して、開発のフロー全体 (価値を提供できるまで) の全体最適を行い、効率の向上させるために現状を把握し、ボトルネックを特定する。
  - **「ベロシティーの計測」**を実施して、チームの進み具合を測定する。効率の継続的な改善のためのベースとなり、また技術的負債やムダによる遅延に気づけるようにする。

価値、阻害要因、課題、プラクティス

次に何をすればよいか
わかり改善できる

フィードバックをうまく活用できない	やり方を評価して改善のためのアクションを見つけることができる	ＫＰＴ法
開発の優先度を正しく設定できない	作ろうとしているものの価値を正しく理解し、ビジネスの成功に結びつかない無駄な開発をすることが無くなる	バックログ
		プロダクト・オーナーの責務と仕事
ボトルネックを特定できない	開発のフローの全体最適を行うことができる	バリュー・ストリーム・マップ
		ベロシティの計測

用語の説明

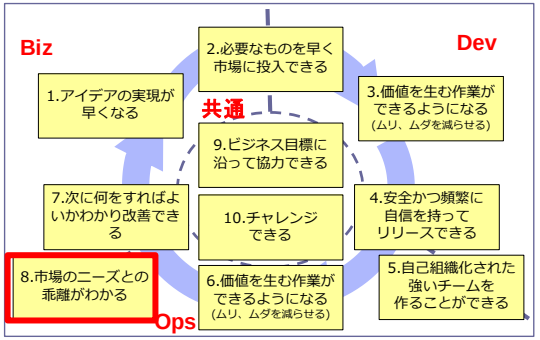
- ＫＰＴ法
  - KPT法は、プロジェクトの振り返りでよく利用される手法です。続けること（Keep）、問題点（Problem）、次にすること（Try）が明らかになり、改善のためのアクションを見つけることが可能となります。
- バックログ
  - 作業優先度が低いなどの理由から、着手せずに放置されている作業です。内外のフィードバックや学習された事実に基づいてバックログの項目を追加または削除し、優先度を更新することにより、無駄な開発をすることが無くなるといった効果があります。
- プロダクト・オーナーの責務と仕事
  - プロダクト・オーナーは、開発が滞らないようにするために、開発すべき機能とその内容について適宜決断を下す必要があります。作ろうとしているものの価値を正しく理解し、ビジネスを成功へ導くことは、プロダクト・オーナーの責務です。
- バリュー・ストリーム・マップ
  - ソフトウェア開発の工程を最適化するため、現在の作業フローを把握し、将来のあるべき姿を明確にするために作成されるプロセス図のことです。特定の製品の開発企画が始まってから、製品がお客様の手に渡るまでの全工程の経路、作業内容、作業時間などを明らかにすることで、作業内に潜むムダを排除し、全体最適化により生産性を高めることができます。
- ベロシティの計測
  - チームの進み具合を測定する作業です。効率の継続的な改善のためのベースとなり、技術的負債やムダによる遅延に気づけるようになります。ボトルネックを特定することにより、開発フローを全体最適化することができます。

8. 市場ニーズとの乖離がわかる

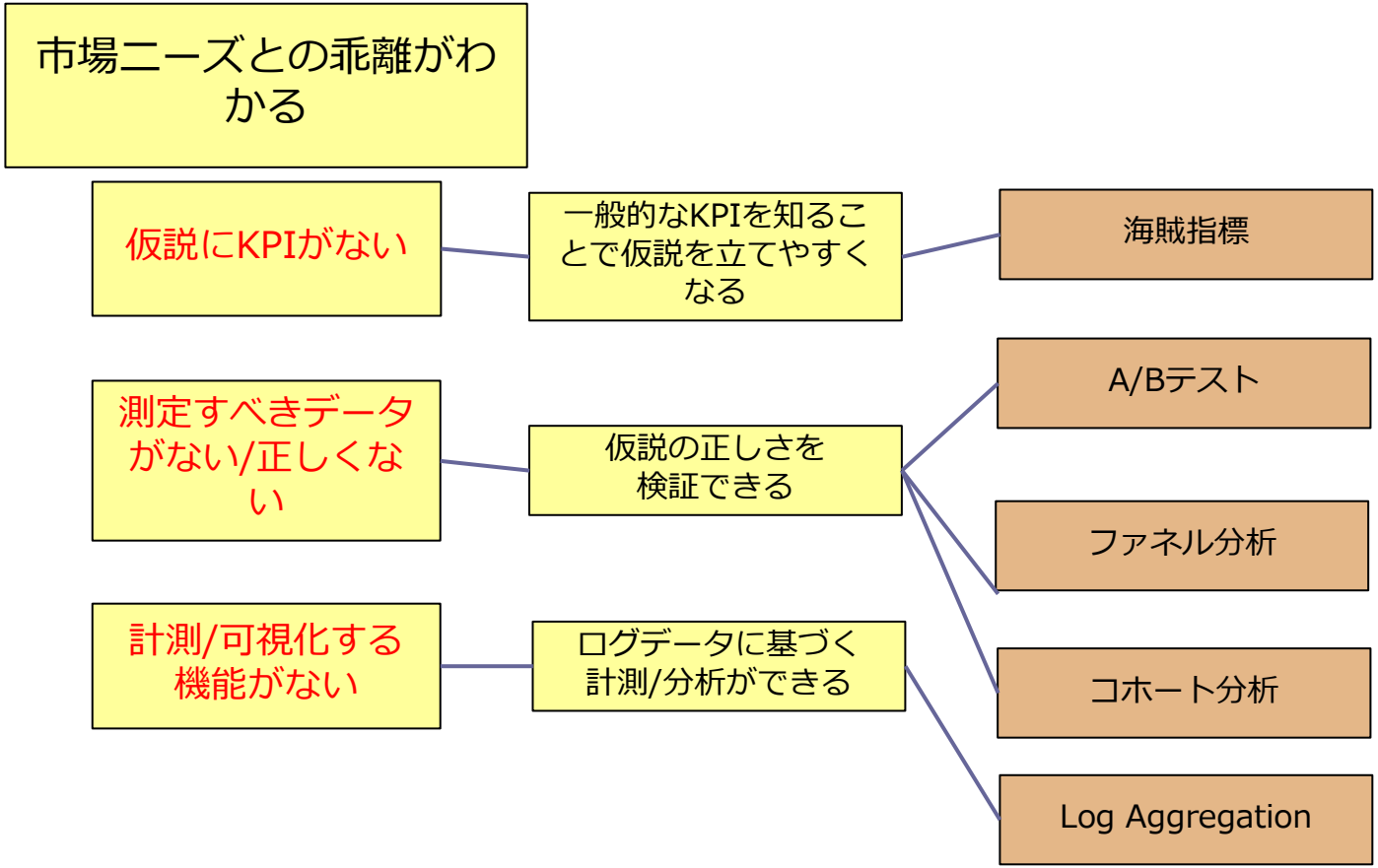
■ 要約
- 早く市場に出してフィードバックが得られれば、仮説の正しさを検証することができ、市場のニーズとのずれを計測することができる

■ 実現を妨げている課題
- 仮説立案時に、「それをどのように検証するか」を決めずに構築してしまった
- 検証方法が実際的に計測不可能だったり、正しい情報が取れていなかった
- 構築時に必要なログを収集するなどの機能を作りこんでいなかった

■ プラクティス
- デイブ・マクルーアの**「海賊指標」**などを参考に、企画の段階から、その仮説の定量的な効果を考えておく
- 定量的な指標は後で収集・分析・可視化ができるように、必要なログを出力できるように開発する。AWSは**「Log Aggregation」**を提唱。
- 現状と仮説の比較を行うための**「A/Bテスト」**や、狙ったターゲットに対して効果的に作用しているかを分析する**「ファネル/コホート分析」**が有効。

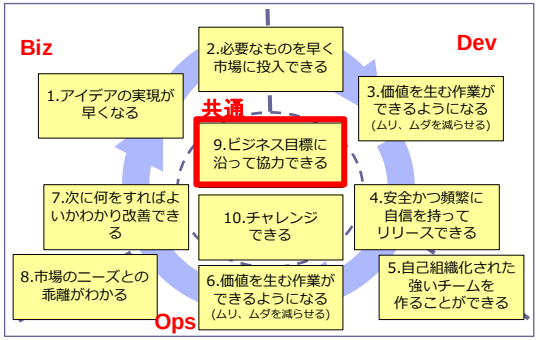


価値、阻害要因、課題、プラクティス

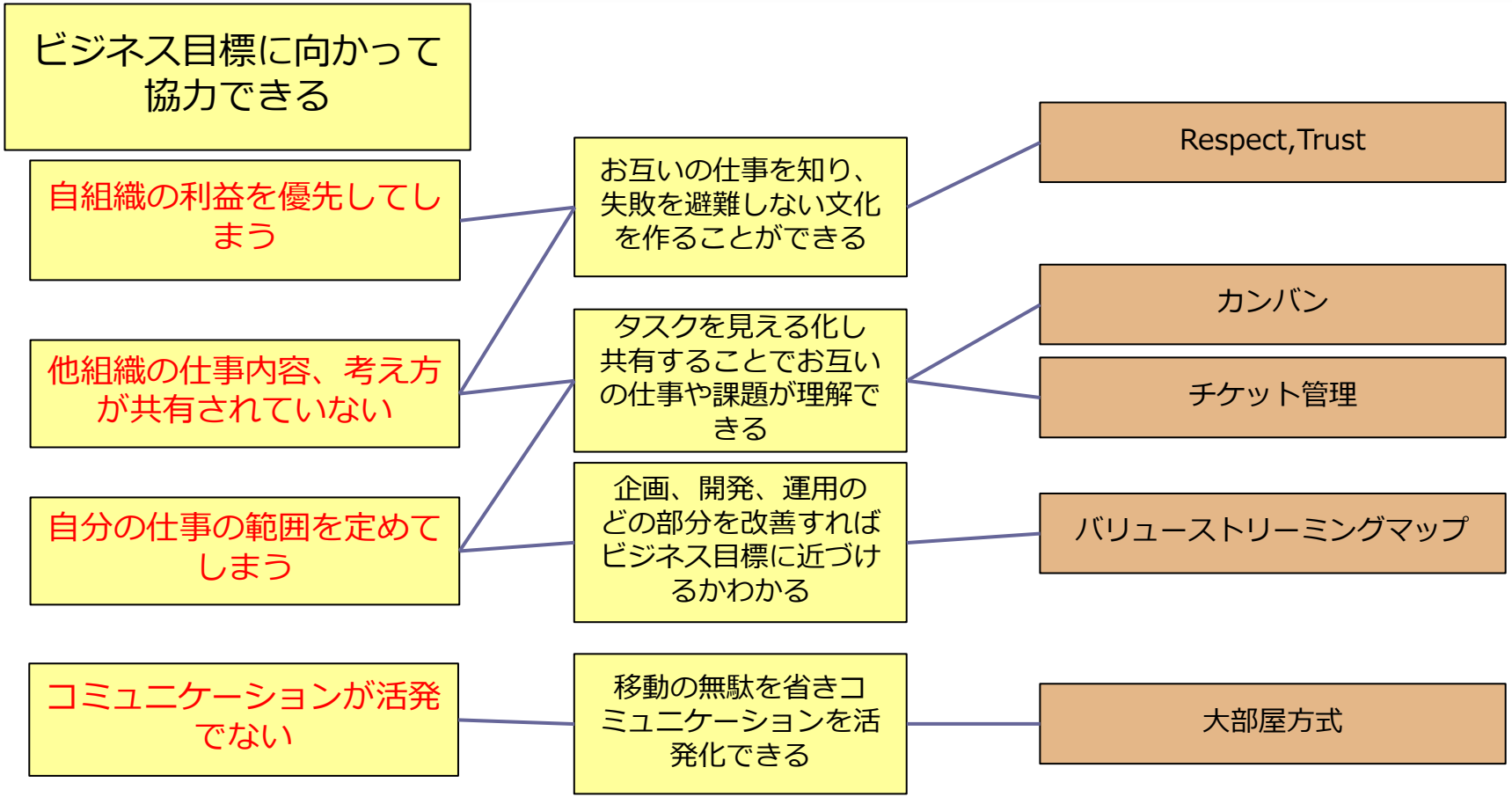


9. ビジネス目標に向かって協力できる

- 要約
  - DevOps チームは、ビジネス目標を共有することで、自組織の都合に囚われず、目標に向かって協力でき、迅速で的確な意思決定、問題解決ができる。
- 従来手法における阻害要因(失敗例)
  - 組織の壁：自組織の利益を優先してしまう
  - 他組織の仕事内容、考え方が共有されていない
  - 自分の仕事の範囲を決めてしまう
  - コミュニケーションが活発でない
- プラクティス
  - **「Respect,Trust」**によりお互いの仕事を知り、失敗に対して建設的な態度を取り非難しない文化を作ることができる
  - **「カンバン」**や**「チケット管理」**によりタスクを見える化し、共有することでお互いの仕事や課題が理解できる
  - **「バリューストリーミングマップ」**により企画、開発、運用のどの部分を改善すればビジネス目標に近づけるかわかる
  - **「大部屋方式」**により物理的な壁をなくし、移動の無駄を省くことでコミュニケーションを活発化できる

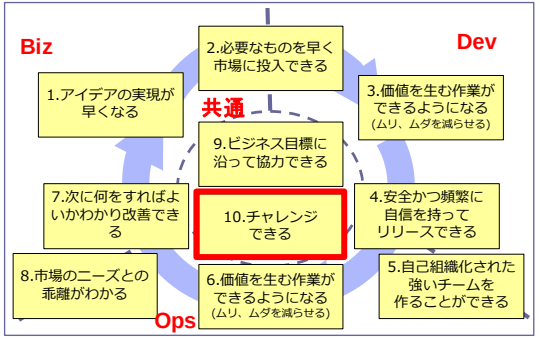


価値、阻害要因、課題、プラクティス

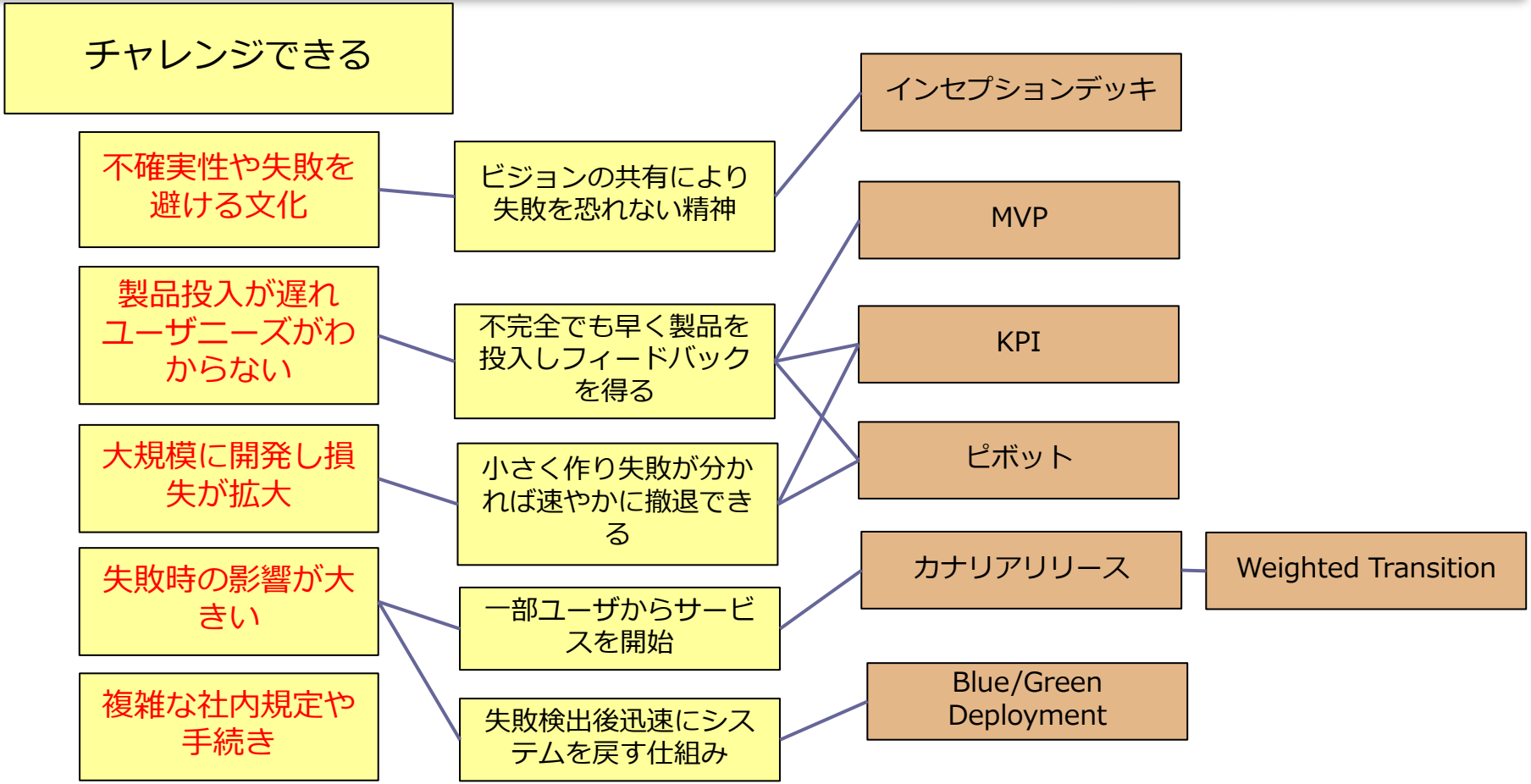


10. チャレンジできる

- 要約
  - 小さく始めて、素早く効果を確認することで、チャレンジを繰り返すことができる
- 実現を妨げている課題
  - (文化)不確実性や失敗を避けようとする文化や企画重視の体質
  - (スピード)最初の製品の市場への投入が遅く、プロジェクト停止の判断が遅れる
  - (コスト)ニーズを見極めずに大規模開発に走り失敗時の損失が拡大する
  - (リスク回避) 失敗した時の影響や元に戻すための時間が見積れず、チャレンジしない方がましと判断してしまう。
  - (リスク回避) 社内手続きの煩雑さや複雑な社内規定に躊躇してしまう
- プラクティス
  - (文化)**「インセプションデッキ」**により、チームのビジョンを共有
  - (スピード)**「MVP」**(Minimum Viable Product)を作り、市場からフィードバックを得て、ニーズの有無を検証する
  - (コスト)**「KPI」**が目標に達しない際は速やかに**「ピボット」**する決断
  - (リスク回避) **「カナリアリリース」**や**「Weighted Transition」**など、一部のユーザからサービスをリリースする仕組み。**「Blue/Greenデプロイメント」**に基づく、リリース前のシステムを保持しながら新規にリリース環境を構築する手法。



価値、阻害要因、課題、プラクティス



IT運用をチカラに

OpsLabo