第26期　ITソリューション塾　特別講座
ソフトウエア・エンジニアリング講座

Agile and DevOps

デジタル・トランスフォーメーション

平成29年11月29日

戸田　孝一郎
株式会社戦略スタッフ・サービス
社団法人TMS&TPS検定協会　理事

デジタル・トランスフォーメーションとは？

- 「ITの浸透が、人々の生活をあらゆる面でより良い方向に変化させる」という概念
- 既存ビジネスをアナログからデジタルへ、デジタルからアナログへとシームレスに変換できる組織への変革
- 全ての企業はITを中心とした組織に変わる
- 全ての組織がITサービス・プロバイダーに変質するという事
- IT（情報技術）が一般化すると優位性はデータを保有する側に移る

デジタルフォーメーションの型

Speed、機敏性を重視
（トヨタ型）

ビジネス・プロセスのスピードを上げて競争力を生むビジネス戦略

イノベーションを重視
（アップル型）

今まで世の中にないイノベーティブな商品/サービスを作り競争力を生むビジネス戦略

あらゆるモノをサービス化
スピードとイノベーションの複合
（アマゾン型）

究極のデジタル化で、上記二種の要素を複合してあらゆる製品、サービスを組み合わせられるサービスを作り出して圧倒的な競争力を生むビジネス戦略

VeriSM™とは？

- Digital Transformationの時代
  - 全ての組織がサービス・プロバイダー化する可能性がある。
  - どの様にITサービスを提供し、維持するのか？
- VeriSM™とは、組織が持っている固有の環境を前提にしたサービスマネジメントからのアプローチ
- 基本的価値観
  - Value-driven (価値主導）
  - Evolving（発展、展開する）
  - Responsive（敏感に反応する）
  - Integrated（統合、結合された）
  - Service（サービス）
  - Management（マネジメント）

対象は、

- 経営者
- 管理者（事業部＆IT）
- サービス・オーナー、サービス・マスター
- IT専門職（DevOpsチーム）

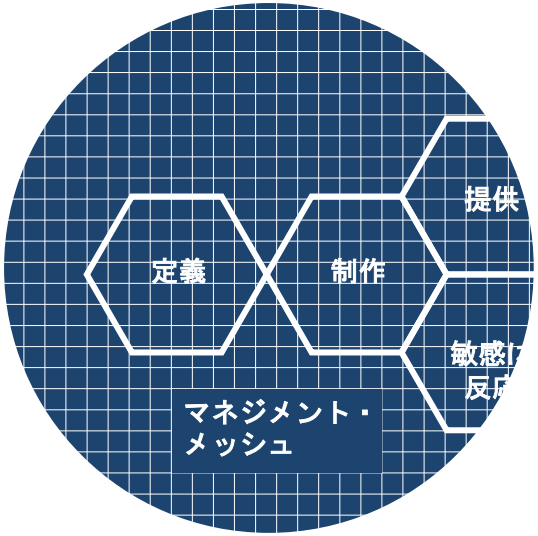
活動

- IFDC (International Foundation for Digital Competences)
- プラクティスの本発行

VeriSM™の基本概念

ガバナンス（企業統治）

サービス・マネジメント指針

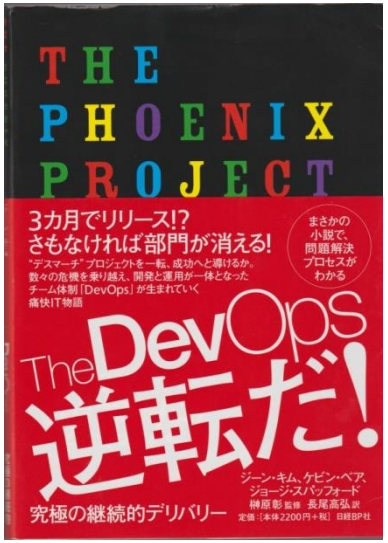


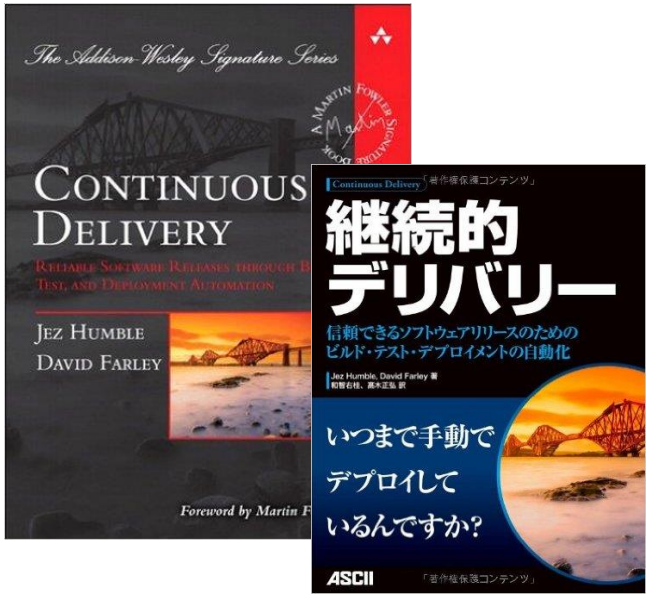
DevOpsとは、

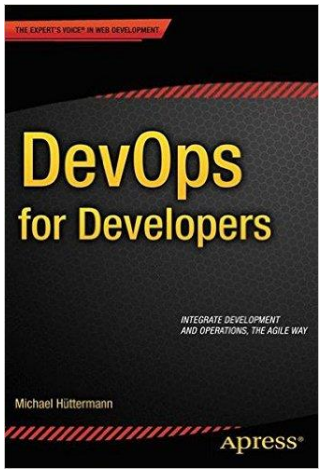
- Development + Operation　の造語
- 2009年に「Velocity 2009」において、flickr のエンジニアにより初めて公の場で用いられた。
  - このプレゼンテーションでは「開発と運用が協力することで、1日に10回以上のペースでリリースが可能になる」という発表とともにDevOpsという単語が用いられた。
- ITコンサルタントのパトリック・デュボア氏は開発と運用を結び付ける手段としてDevOpsを提唱し、2009年に最初のDevOpsカンファレンス「DevOpsDays」を開催している。
- アジャイルとリーンの原則がソフトウエア・サプライチェーン全体に適用される。
- ALM（アプリケーション・ライフサイクル）の視点での管理プロセス
- ビジネス・プロセスとしての側面
- DevOpsとは、人、文化
  - DevOps is not Tool, DevOps is People, People use Tool. (Carnegie Mellon SEI)

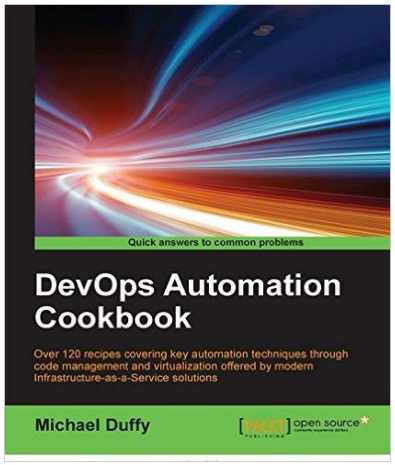
JITでのITサービスの提供

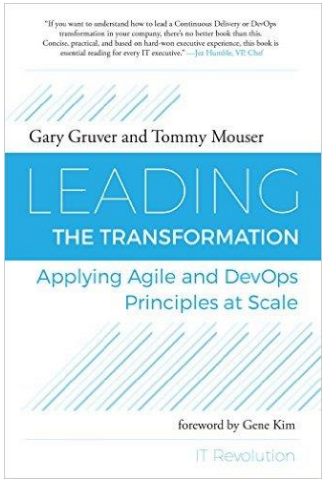
企業におけるITの価値の本質に立ち返る活動
ITの価値の本質　＝　ビジネスをタイムリーに支援する。
ビジネス・スピードを牽引する。
ビジネス領域を拡大させる。
見えなかったモノを見える様にする。

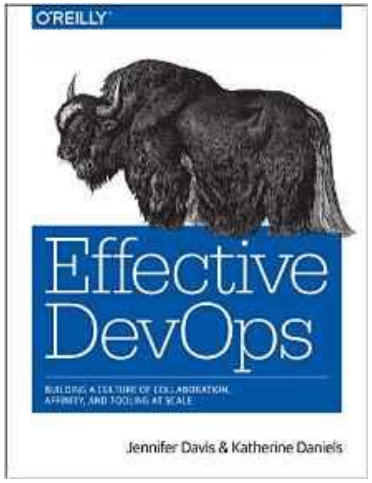












運用系でも、アジャイル＆DevOps
ITILの進化

プロセス化

ITIL V1. (1986年～)　：　ITサービスの利用と提供のガイドライン
業務中心

ITIL V2. (2000年～)　：　単体業務からプロセス（プラクティス）で体系化と業務の均一化
①プロセス・アプローチとベスト・プラクティス
②IT技術とビジネス視点で、7つの領域に整理して
サポート（青本）とデリバリー（赤本）を定義

ITIL V3. (2007年～)　：　オープン化への対応とビジネスの継続性（BCP）重視
①ビジネスとITの統合
ビジネス価値を意識
②バリューネットワークの概念を導入
ビジネス視点を原則としALMの実践
DevOps（アジャイル開発）
ⅰ アプリケーション開発（調達）
要件定義、設計、構築
ⅱ サービス・マネジメント
展開、運用、最適化、自動化
ⅲ アプリケーション・ポートフォリオ

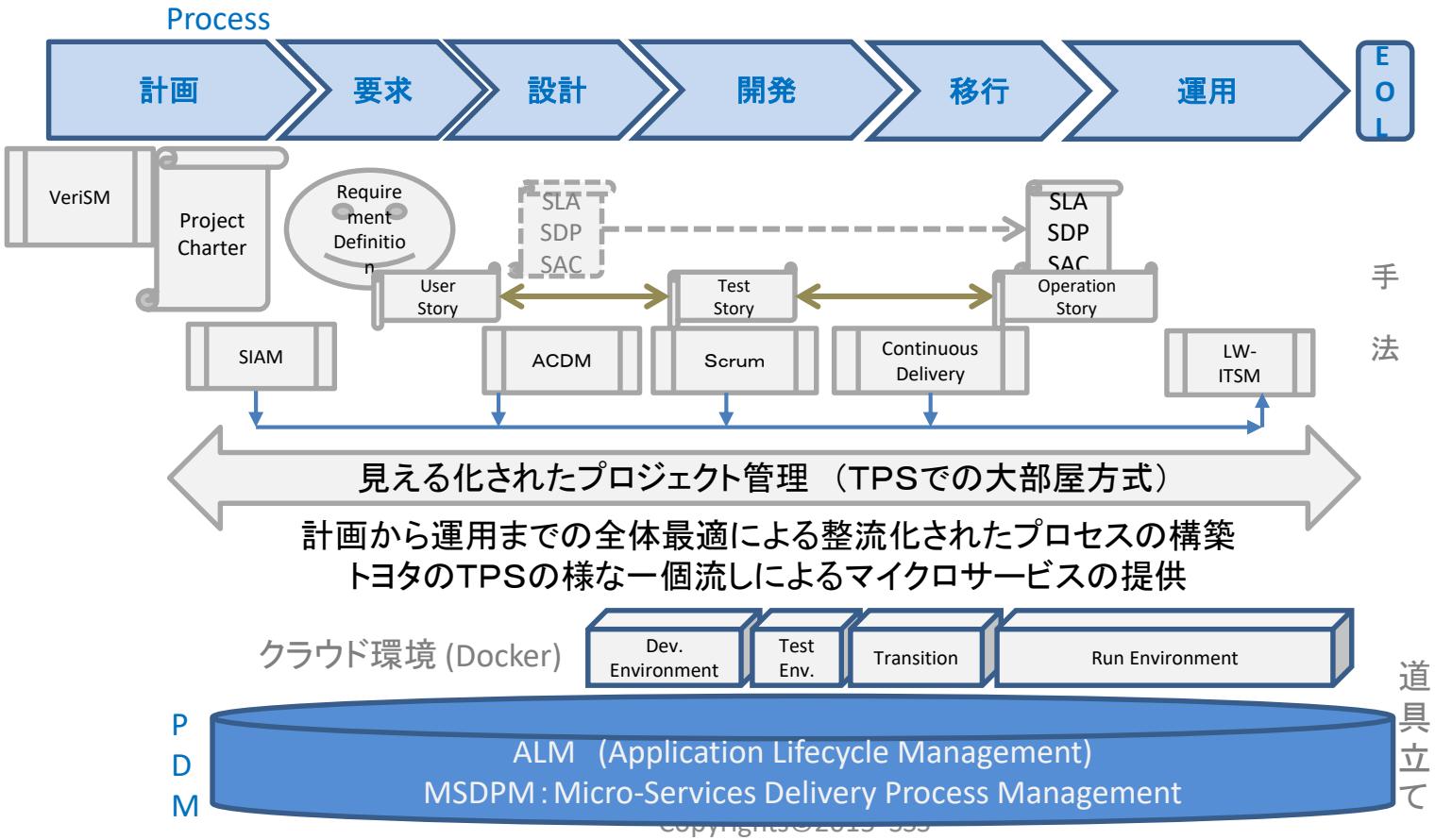
バッチ・プロセス
から
一個流しプロセス
へ

③SMO（Service Management Office）← 2011版で追加
プロセスの成熟度を上げる目的
プロセスの構築とプロセスの成熟度を上げる活動を明確に分ける
（CMMIで言うPMOと同様）

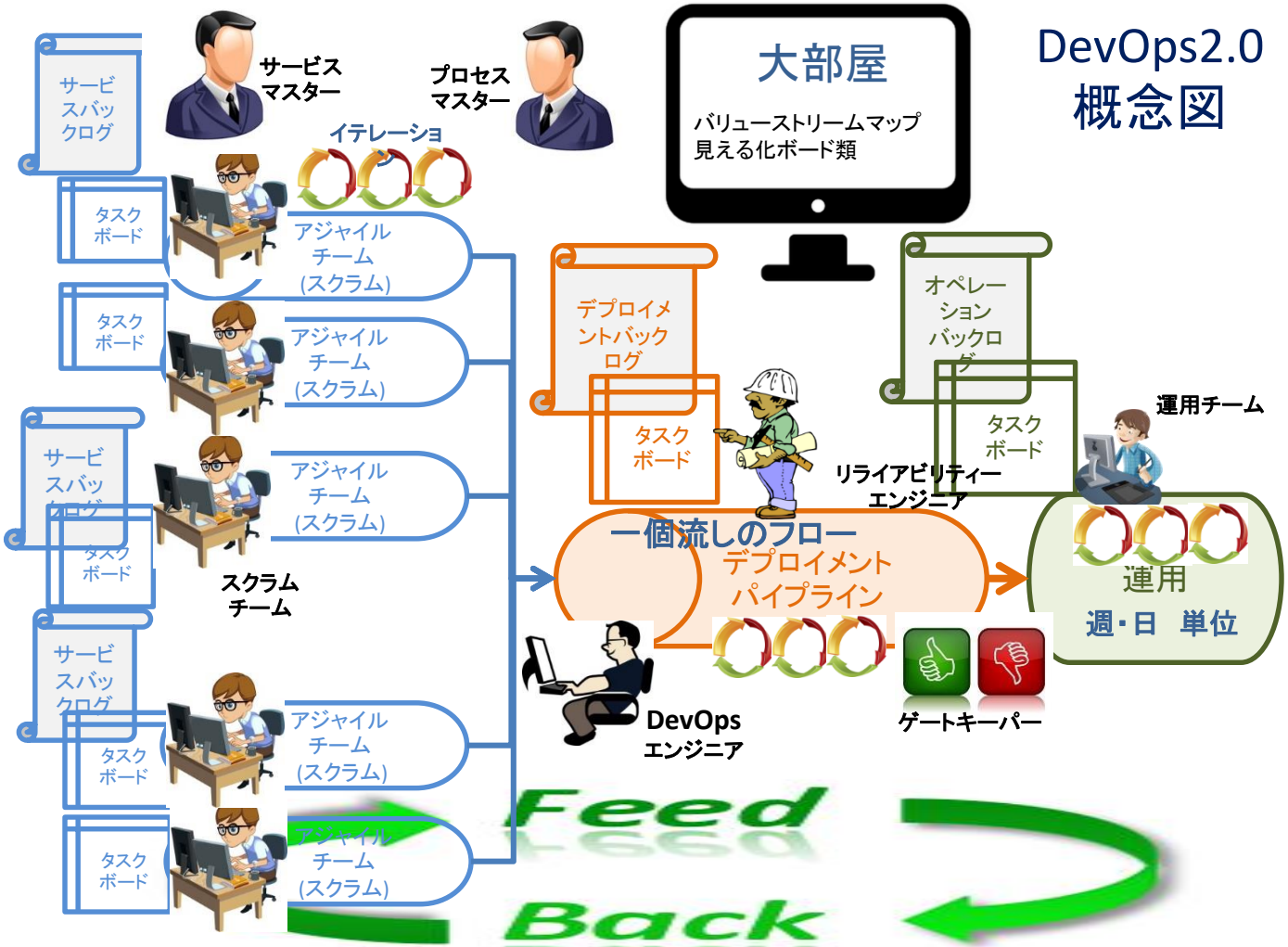
DevOpsの最終的なゴール

縦割りの企業情報システム & ITプロセスの仕組み

エンタープライズDevOps概念図

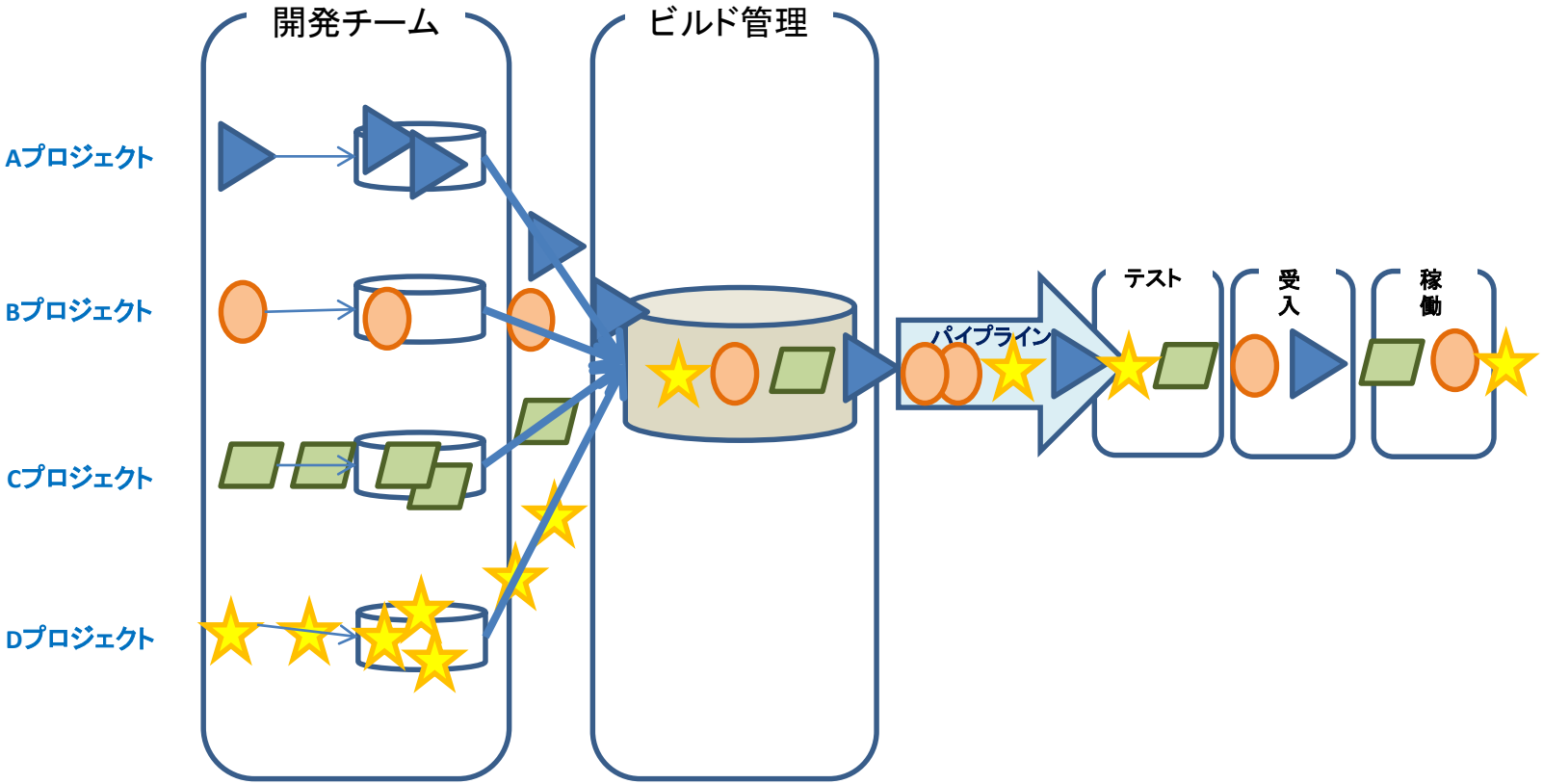


DevOps2.0 概念図

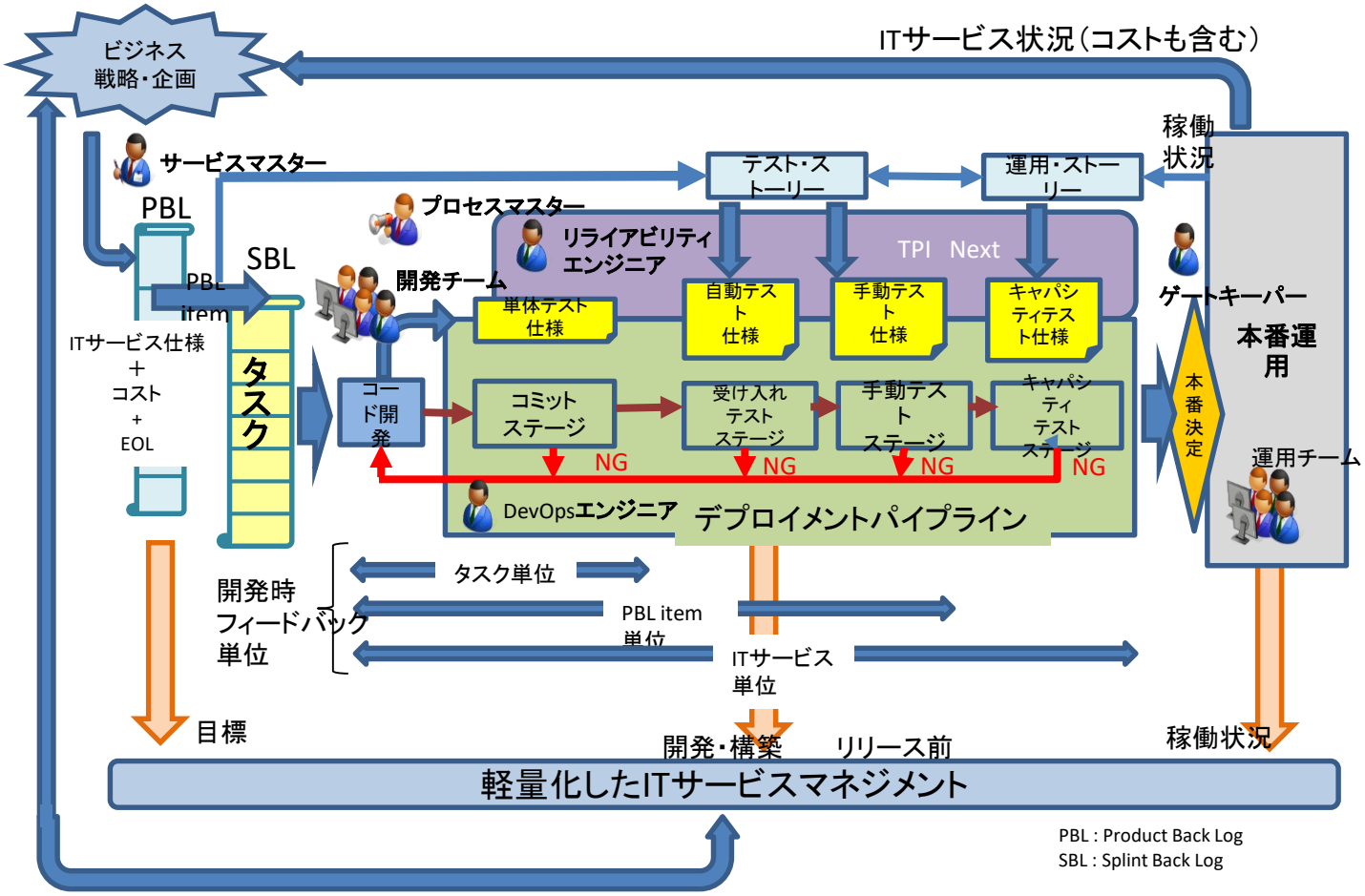


計画から運用までの全体最適による流水化されたプロセスの構築

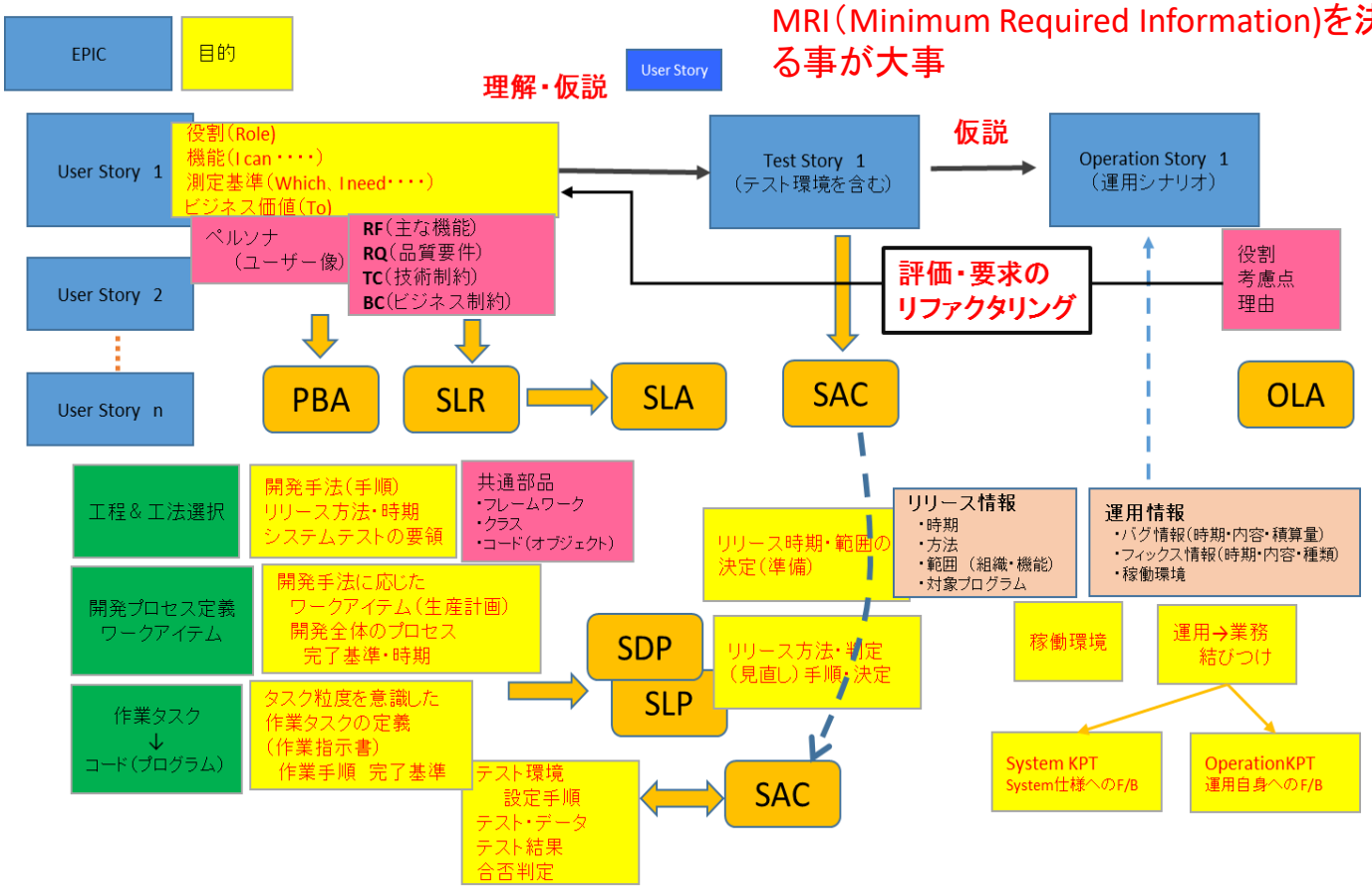
DevOpsのプロセス（模式図）



デプロイメントパイプラインの概略フロー



軽量化されたITサービスマネジメントを適用した全体概要図



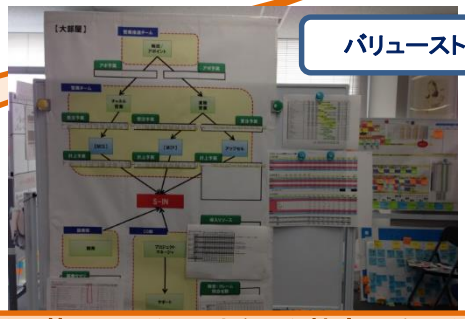
各作業現場では、正しく作業を行うために、それなりの情報が創出されている。

DevOps2.0の事例

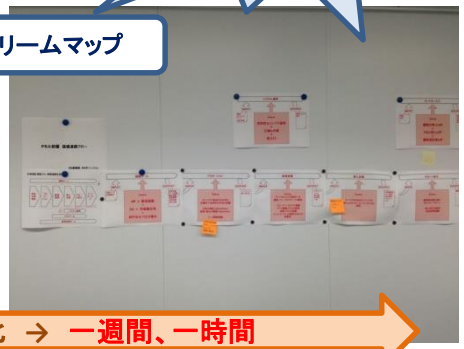
2009年　　アジャイル開発を開始
2011年秋　　アジャイル開発は成功裏に導入できたが、、、
　　課題噴出：　開発工程はボトルネックでは無かった。
2012年4月　　ビジネスプロセスの全工程の見直しと整流化のプロジェクト開始
2013年10月DevOpsの導入（全プロセスの見える化、同期化と大部屋化実現）

事業規模が
2年で3倍以上

非製造業で初の
大部屋の実現



バリューストリームマップ

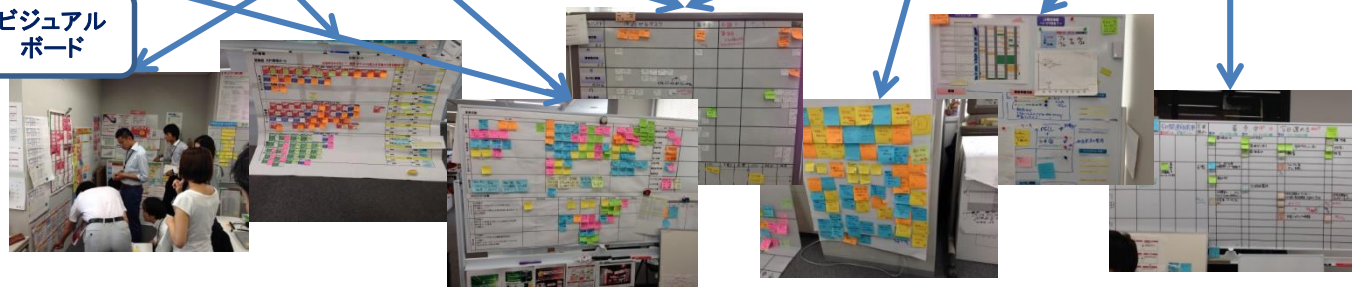


xx事業部ビジネスプロセス

プロセスの同期化（管理サイクル）とタスク粒度の均一化　→　一週間、一時間

企画　営業　管理　デザイン　開発　移行　運用　コールセンター

ビジュアルボード



ITシステムの開発手法

【Waterfall Development】

ソフトウエアを作成することが目的
評価は、計画通りに完成する事
19世紀のフレデリック・テイラー（IEの創始者）の「科学的管理法」が基本思想

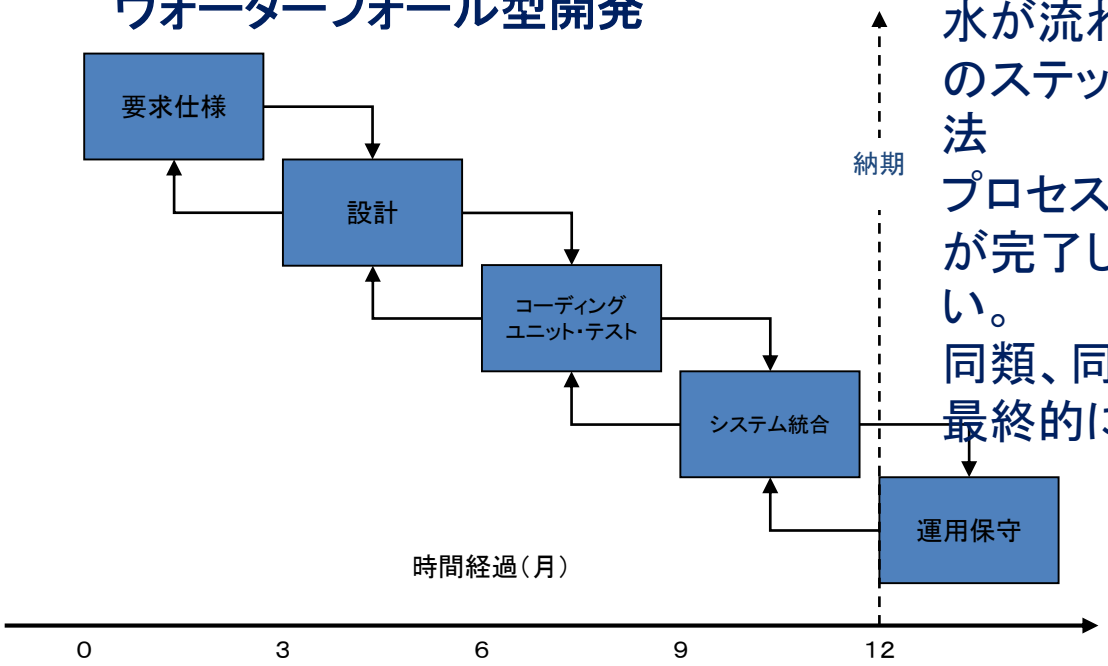
【Agile Development】

ITサービスを実装し、提供することが目的
評価は、ビジネスの成果
フェレデリック・テイラーの考え方に疑問？を唱える(Kent Beck)

問題となる三つの単純化された仮定
① 通常、物事は計画通りに進む
② 局所最適は全体最適に繋がる
③ 人はほぼ代替可能であり、何をすべきかを指示する必要がある。

開発手法の違い（概要）

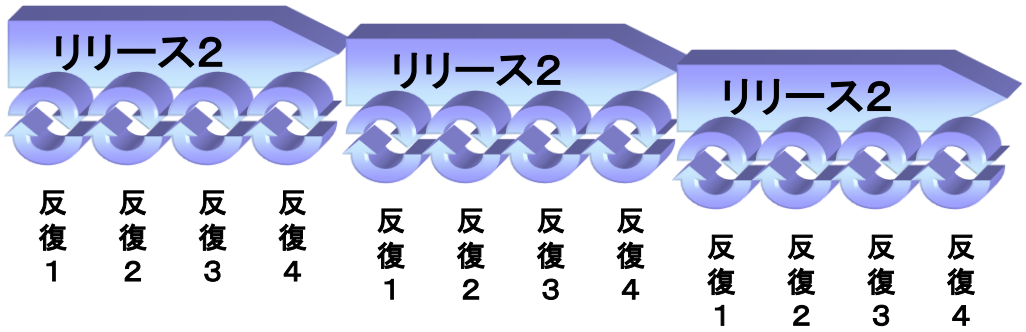
ウォーターフォール型開発



水が流れ落ちる様にプロジェクトのプロセスのステップを順を追って作業を進めていく方法
プロセスの各ステップで必要な全ての作業が完了しなければ、次のステップに移行しない。
同類、同質な作業を貯めて、まとめて行う。
最終的にリリースは、一回で実行される。

アジャイル型開発

期間を決めて、その期間内で小さな稼働可能な部分のリリースを小まめに実施し、フィードバックを得て修正を加えつつユーザーの望むシステムを完成させる方法



各反復期間内では、開発チームによって、設計、コーディング、単体テスト、結合テストが実施され、常に稼働可能な状態で段階的にリリースを繰り返す。

何を変えるべきか？

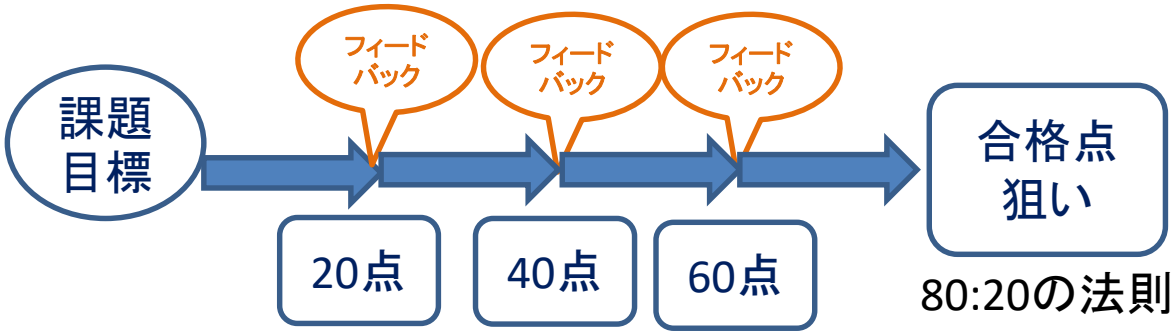
	ウォーターフォールを中心とした 従来の方式	アジャイル、DevOpsの 新しい方式
価値観	計画重視	リソース重視、適応性重視
スピード	遅い	早い
働き方	仕事はまとめた方が効率が良い 残業を認める	仕事は一つづつこなした方が効率が良い 残業を認めない （定時間労働）
時間の使い方	仕事の目的を達するまでの時間	区切られた時間内で仕事の目的を達成 →タイムボックス
計画	計画重視 全期間にわたる計画立案 （計画通りにコトは運ぶ）	計画作り重視（朝令暮改） 当面1ヶ月は詳細に、後は粗い計画 （計画通りにはコトは運ばない）
プロジェクト	しっかりと計画を立て、理論的に 進める。	反復的（イタラティブ）に進める フィードバックが重要
リスク	プロジェクト後半に増大	プロジェクト前半に集中
役割	専門家（分業化）	多能工化（T型人材））
進捗管理	管理指標	現地現物管理（働くプログラムのみ）
見える化	作業が終わらないと見えない	ほぼ何時でも見える化（透明性）
評価基準	計画に対して	ビジネス価値（ビジネスの成果）
要求	管理可能、100%定義可能	管理不能、定義不能（標的は動く）
設計	機能中心設計	プロセス中心設計
開発	基本は個人（専門家）	チームの連帯責任
コード	属人化する	属人性の排除
品質	管理強化（当たり前品質）	向上可能性あり（魅力的品質）
ドキュメント	多種多様、管理基準	MRI （最低必要限）使用目的の明確化
デプロイ＆リリース	半自動	自動化 （デプロイメント・パイプライン）
運用	分権独立	オペレーションの一体化
運用管理	ITIL	軽量化したITサービス管理＆ISO20000
リーダーシップ	統率型（指示＆命令）	サーバントリーダーシップ（ファシリテーション）

アジャイル開発の仕事の基本構造

イテレーション（スプリント）
何故、反復的に開発を行った方が良いのか？
何故、そうしなければならないのか？

【従来の考え方】

【アジャイルの考え方】

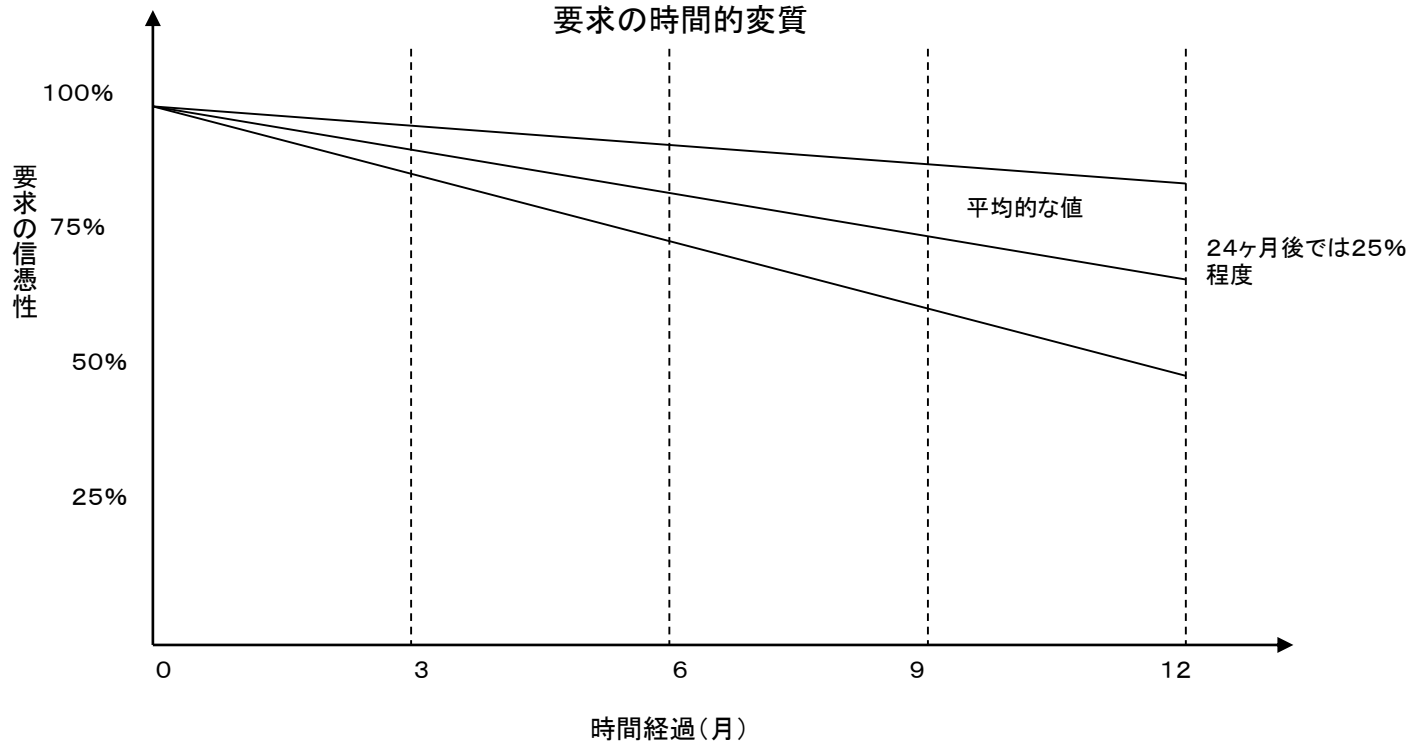


80:20の法則

- **時間の使い方 （タイムボックス）**
- **透明性**
- **品質**

要求の変質

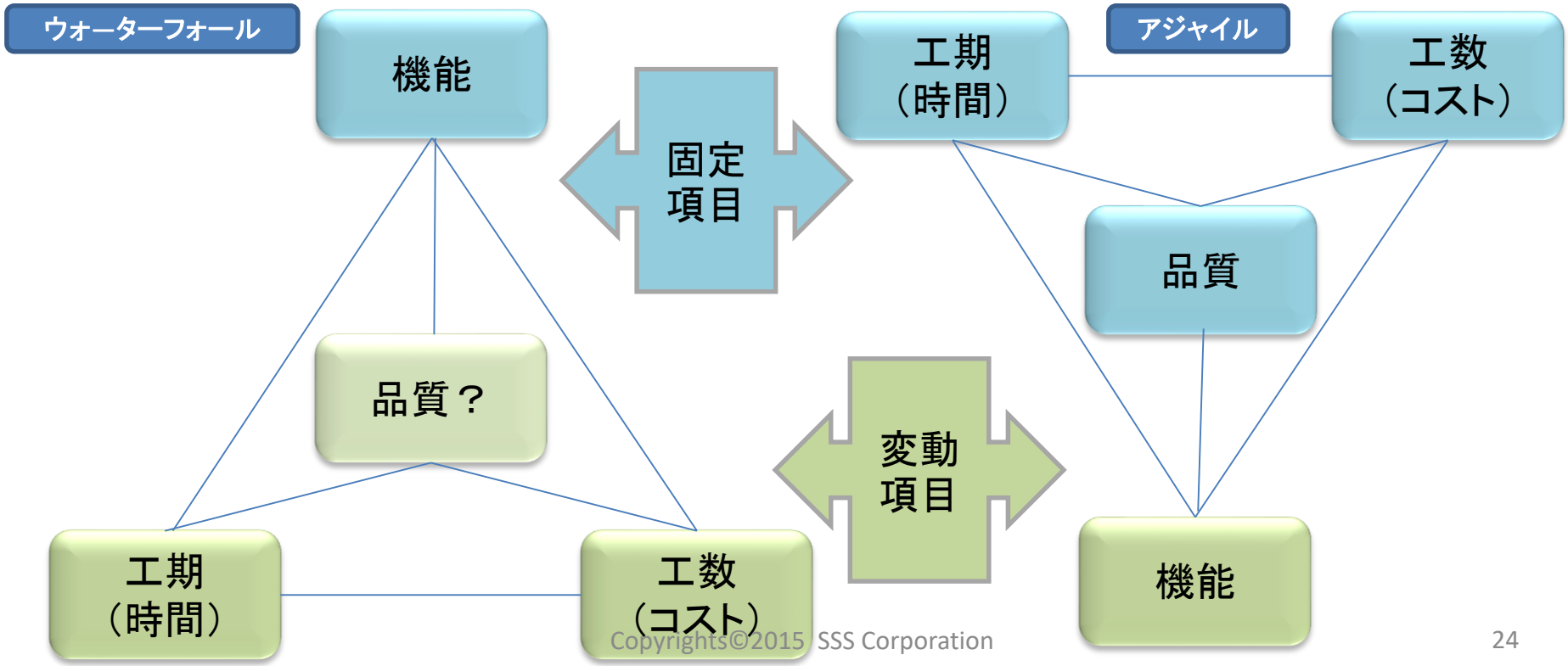
要求を明確に定義できれば良いシステムができるという迷信



アジャイル開発の基本行動

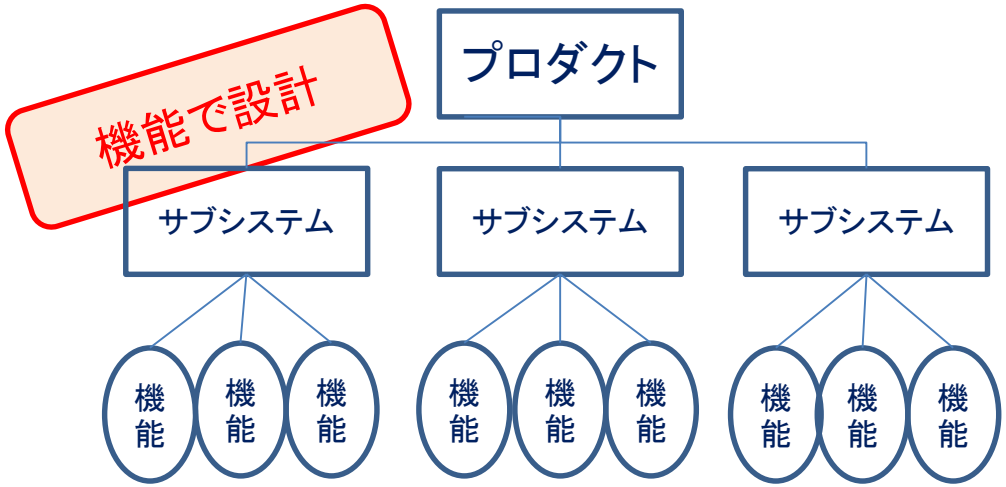
基本的な価値観：

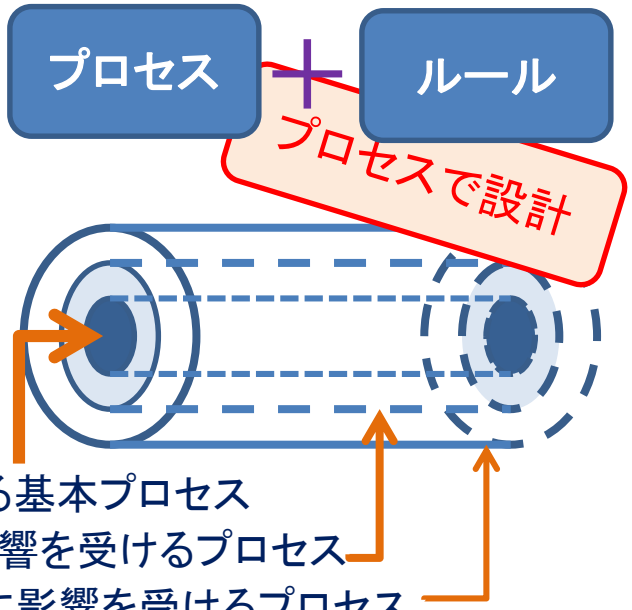
与えられたリソース（資源）の制約下で、最大のパフォーマンスを発揮する。
常にカイゼンを行い、作業効率の向上を目指す。
タイムボックスでの働き方で、時間の有効活用
反復（イタレーション）を利用して、こまめにフィードバックを得て、軌道修正を常に行う。
ゴール（標的）は、固定されていない。常に動く。
常に動作するソフトウエアで確認する。
品質を犠牲にしない。品質を高め、維持する努力。



システム設計方法の変化

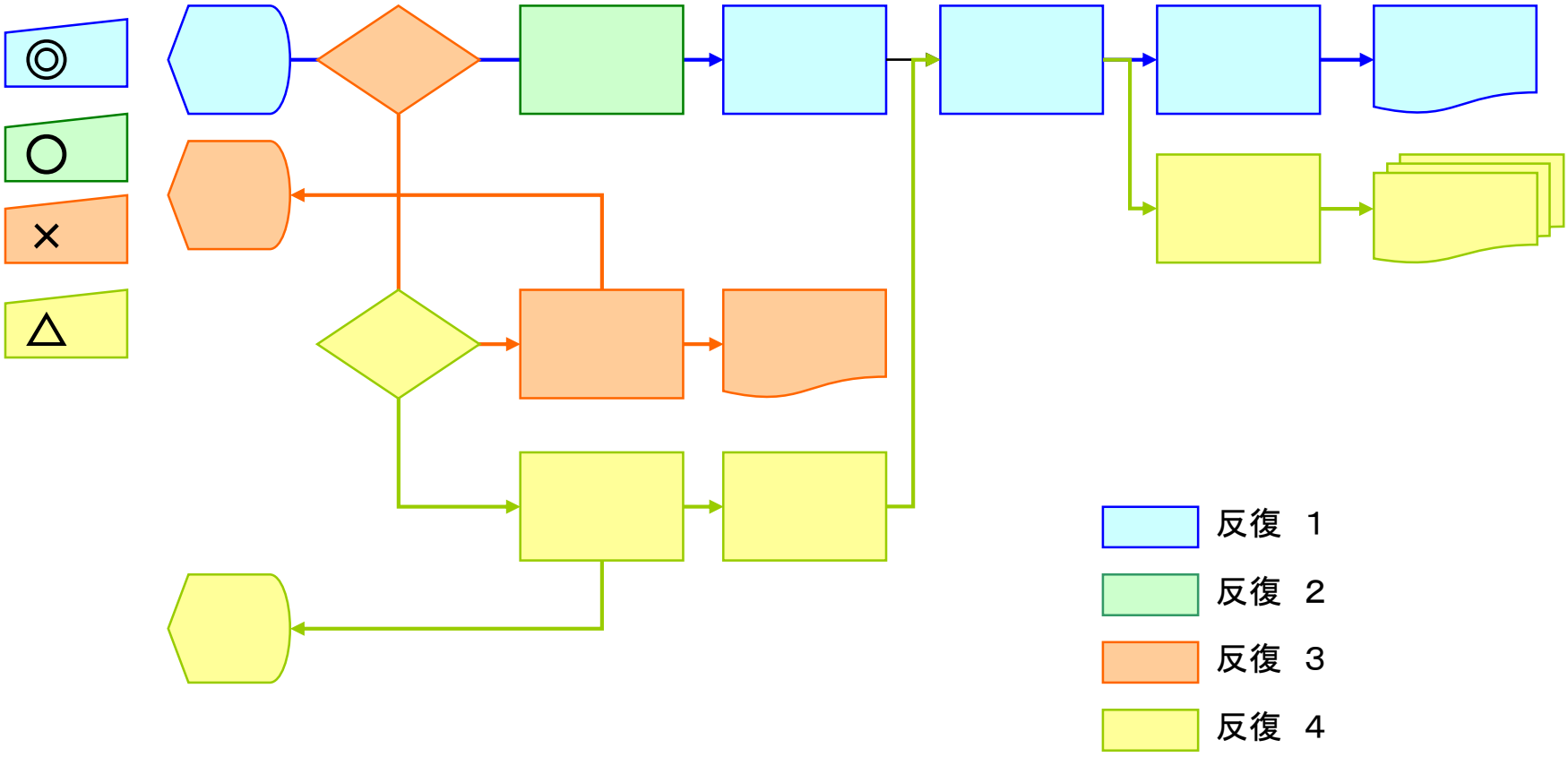
- 何故オブジェクト(SOA)を使用しているのに再利用が困難なのか？再利用されないのか？
  - ソフトウエアの標準部品(コンポーネント)は簡単に設計できるのか？
  - いきなり標準部品(コンポーネント)の設計に入って居ませんか？





- **業務プロセスで設計するという方法があります。**
  - ユーザー・ストーリー
  - ボディー・プロセス(Body)： 業種を問わず共通している基本プロセス
  - オルタナティブ・プロセス(Alternative)： 業種特性に影響を受けるプロセス
  - オプション・プロセス(Option)： 企業の商慣習や慣行に影響を受けるプロセス

アジャイル開発におけるソフトウエアの作り方



アジャイル開発―概論

高品質でムダの無い、且つ変更要求に対応できるソフトウエアを作成する為の適切な一連の手順に従い高い協調と自律的なプロジェクト関係者によって実施される反復（周期）的、インクリメンタルなアプローチである。

■ダイナミックシステムズ開発技法（Dynamic Systems Development Method）
デイン・フォルナー（Dane Faulkner）ほか
■アダプティブソフトウェア開発（Adaptive Software Development）
ジム・ハイスミス（Jim Highsmith）
■クリスタルメソッド（Crystal Methods）
アリスター・コックバーン（Alistair Cockburn）
■スクラム(Scrum)
ケン・シュエイバー（Ken Schwaber）、ジェフ・サザーランド（Jeff Sutherland）
■ XP（エクストリームプログラミング）
ケント・ベック（Kent Beck）、エリック・ガンマ（Eric Gamma）ほか
■リーンソフトウェア開発（Lean Software Development）
トムおよびメアリー・ポッペンディーク（Tom and Mary Poppendieck）
■フィーチャ駆動開発（Feature-Driven Development）
ピーター・コード（Peter Code）、ジェフ・デルーカ（Jeff DeLuca）
■アジャイル統一プロセス（Agile Unified Process）
スコット・アンブラー（Scott Ambler）

- 反復（周期）的(Iterative) --- 定期的なリリース
- 漸進的(Incremental) --- 徐々に機能を増加
- 適応主義(Adaptive) --- 変化に対応（即応）
- 自律的(Self-Organized) --- 学習する組織
- 多能工(Cell Production) --- 一人多役（SE、プログラマー、テスター）

アジャイル・マニフェスト2001

- アジャイル・ソフトウェア開発宣言

  我々は、自らアジャイル開発を実践するとともに、
  人々がアジャイル開発を実践するための支援を通じて、
  より優れたソフトウェア開発方法を見つけようとしている。
  この活動を通じて、我々は、

人と人同士の相互作用を、	プロセスやツールよりも
動くソフトウェアを、	包括的なドキュメントよりも
顧客との協力を、	契約交渉よりも
変化に対応することを、	計画に従うことよりも

  尊重するに至った。
  これは、右側にある項目の価値を認めつつも、
  左側にある項目の価値をより一層重視する、ということである。

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highamith
Andrew Hunt
Rin Jeffries
Lon Ker
Brian Marick
Robert C.Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

http://agilemanifesto.org/

マニフェストの思想を支える重要な方針（アジャイル原理）

1. 我々の最優先事項は、素早いそして継続的な価値あるソフトウエアの提供を通して顧客の満足を得る事である。
2. 開発局面の後半であっても要求の変更を歓迎する。アジャイルなプロセスを顧客の競争優位の為の変化に利用する。
3. 稼動するソフトウエアをより短かい期間を優先して、数週間から数ヶ月で定期的に提供する。
4. プロジェクト期間を通して業務ユーザーと開発者は共同して作業をしなければならない。
5. やる気のある人々を集めてプロジェクトを組織し、彼らが必要とする環境と支援を与え、仕事が完了するまで信頼する。
6. 開発チーム内あるいは開発チームに対するコミュニケーションで最も効率的かつ効果的な手法は、フェイスツーフェイスの会話である。
7. ソフトウエアが正常に機能するということが進捗の基本的な評価である。
8. アジャイルプロセスは持続可能な開発を促進する。スポンサー、開発者、ユーザーは無期限かつ不断に保守できるようにしなければならない。
9. 技術的に優れた良い設計に継続的に配慮する事は機敏性（アジリティー）を増長させる。
10. 簡素が基本　ーやらない仕事をできるだけ多くする
11. 最良の構想（アーキテクチャ）、要求仕様、設計は自己統制された（自律的）チームより出現する。
12. 定期的にチームは振り返りを行い、より効果的に出来る方法を思案し、それに基づいてチームの行動に協調と調整が働く。

スクラム・プロセス

プロダクトオーナーr
（お客様）

プロダクトバックログ
1.----------
2.----------
3.----------
4.----------
5.----------

確実な優先順位付け
（同位は無し）

追加、修正、変更可能
（Sprintに入っている物
以外）

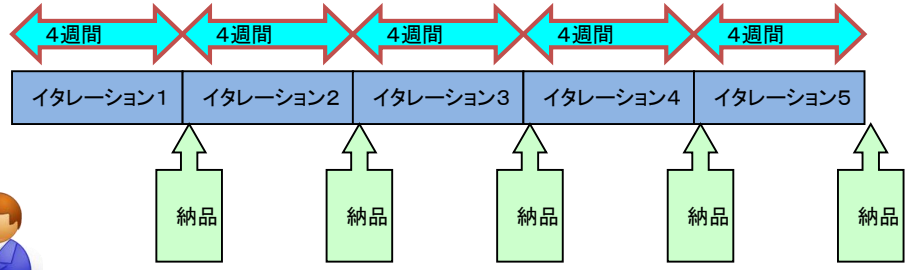
Product Ownerが全て
責任を持つ

スプリントプランニング

パート１
２時間

パート2
２時間

スクラムマスター
（ファシリテーター）



チーム
開発者

デザイン・開発・テスト

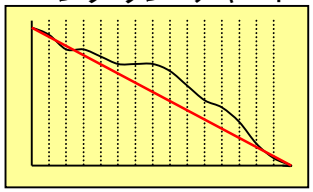
毎朝15分のスタンド
アップ・ミーティング

タスクリスト
(スプリントバックログ)
a.----------
b.----------
c.----------
d.----------
e.----------
f.----------

デイリー
スクラム

スプリントレビュー
３時間

バーンダウン・チャート



レトロスペクティブ
（振り返り）
３時間

タスクボード

To Do	On Going	Done
c, d	b	a
e, h	g	f

スプリント（反復期間）

１週間～４週間

スクラム （Scrum）

- Ken Schwaber & Jeff Sutherland （1995年）
- 特徴
  - 軽量
  - 理解しやすい（シンプル）
  - でも、会得するのは難しい
- 三本の柱
  - Transparency （透明性）
  - Inspection （視察、検査）
  - Adaptation （適応、順応、改作）
- 基本的考え方
  - タイムボックス（Time Box）
    - 時間を区切って、その時間内に目的を果たす仕事を行う仕事の仕方
  - 機能横断的な固定化されたチーム
    - チーム内で役割分担を決めず、全員が必要な仕事をこなす多能工（T型人間のチームでできるだけチームメンバーを固定化した方が良い
  - 持続可能なペース
    - 徹夜や残業を排除して安定した持続可能な一定のペースで開発し、定期的にリリースを行う

（参考）アジャイル開発におけるタイムボックスの価値
＝　やる気と集中力

『仕事の量は、完成の為に与えられた時間を全て使い切るまで膨張する』
イギリスの歴史学者・経済学者であるパーキンソンの言葉

時間には弾力性がある。
時間は、何となく使ったのではいくら有っても足りない。
同じ仕事量でも、意識の違いでかかる時間は全く異なる。

生産性はやる気と集中力で高まる。
やる気のホルモン　＝　ドーパミン
ドーパミンは、ご褒美によって放出される。
やる気はご褒美の事を考えるだけで出る。しかし、裏切られると一瞬で低下する。
ご褒美の60秒ルール　＝　ご褒美は直ぐに貰える事が重要。楽しく想像できる事が重要。

スピードを上げるほど、脳は活性化する。
集中していればミスは少ない。
時間を計ればムダに気づく事ができる。
集中力は長く続かない。休む事で充電される。
時間が読めるからリラックスできる。

『やる気と集中力の高め方』 東京大学　医学博士　森田敏宏著より抜粋

開発チーム

- 自己組織的なチームである
  - 自律性
    - メンバーが自ら日々の仕事を管理する
  - 自己超越
    - 常に目標を達成するように自らを追い込み、常に自身のプラクティスを改善する
  - 相互交換作用
    - 機能横断的かつ定めた役割が無い
- 目標にコミットする義務がある
  - チームはスプリント計画ミーティングでコミットした目標を達成する責任を持つ
- 目標達成につながるならば方法を限定しない
  - チームはすべての決断を下す権限、必要なことを何でも行う権限、あらゆる障害の除去を依頼する権限を持つ
- 争議はチーム内で解決する
- **作業規約**を作る

プロダクト・オーナー

【ミッションと責任】

- プロダクトの完成図と方向性を示す
- プロダクトに必要な機能の詳細を決める
- リリースの内容と日程を決める
- 市場価値に基づくプロダクトバックロイグの優先順位を決める
- スプリント毎に優先順位を変更できる権限を持つ
- プロダクトの収益性（ROI）の責任を持つ

【プロダクトオーナーが行う事】

- プロダクトのビジョンを作成し、関係者に示す。（共有する）
- 対象プロダクトのビジネス要求（ビジネス環境）の説明
- エピック、ユーザーストーリーの確定とペルソナの作成
- ユーザーストーリー毎の受け入れ基準の承認
- プロダクトバックログの優先順位付けとプロジェクト期間中の維持管理
- 開発チームへのユーザーストーリーの説明
- 開発チームのDoD（完了の定義）作成の支援
- リリース計画の承認
- 稼働環境の定義
- EOL（プロダクトの終焉）条件の設定

スクラムの管理者（スクラムマスターの仕事）

◆チームの機能や効率化を支援する
◆チームが最大のパフォーマンスを発揮できる環境を作る
（妨害からチームを守る）
◆チームがスクラム・プロセス通りに作業を実施できる様に支援する

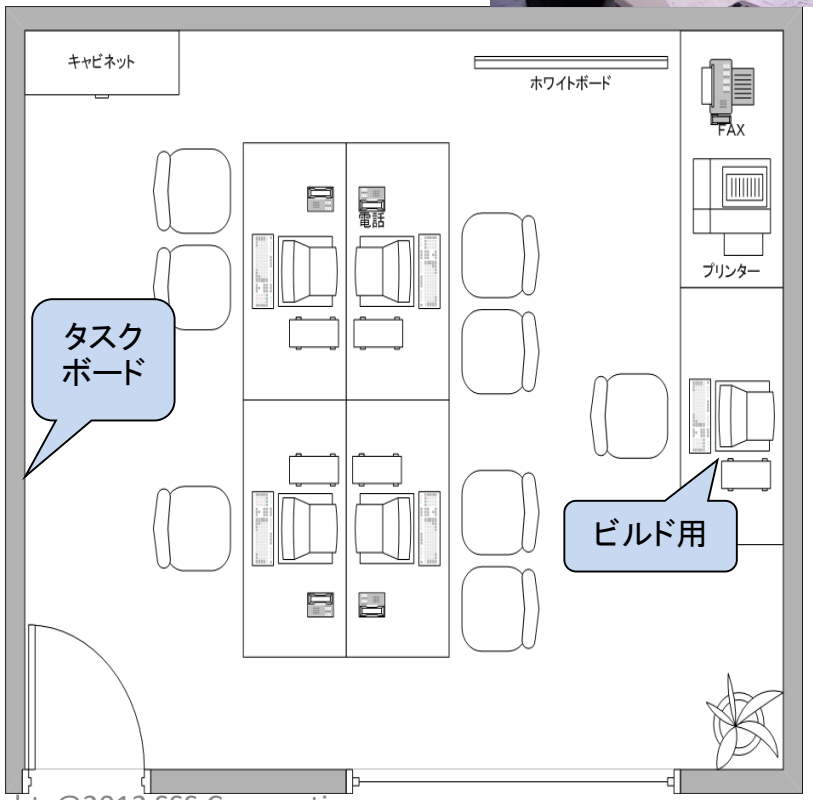
◆チームに気付きを与える
◆チームの自律を支援する
◆チームの能力をユーザーに売り込む
◆プロジェクト関係者（ステーク・ホルダー）間の信頼感を醸成する

◆お客様（ユーザー）第一の思想
◆ジャスト・イン・タイムの徹底
◆カイゼン活動の促進
◆チームのモチベーションの向上

スクラムの環境

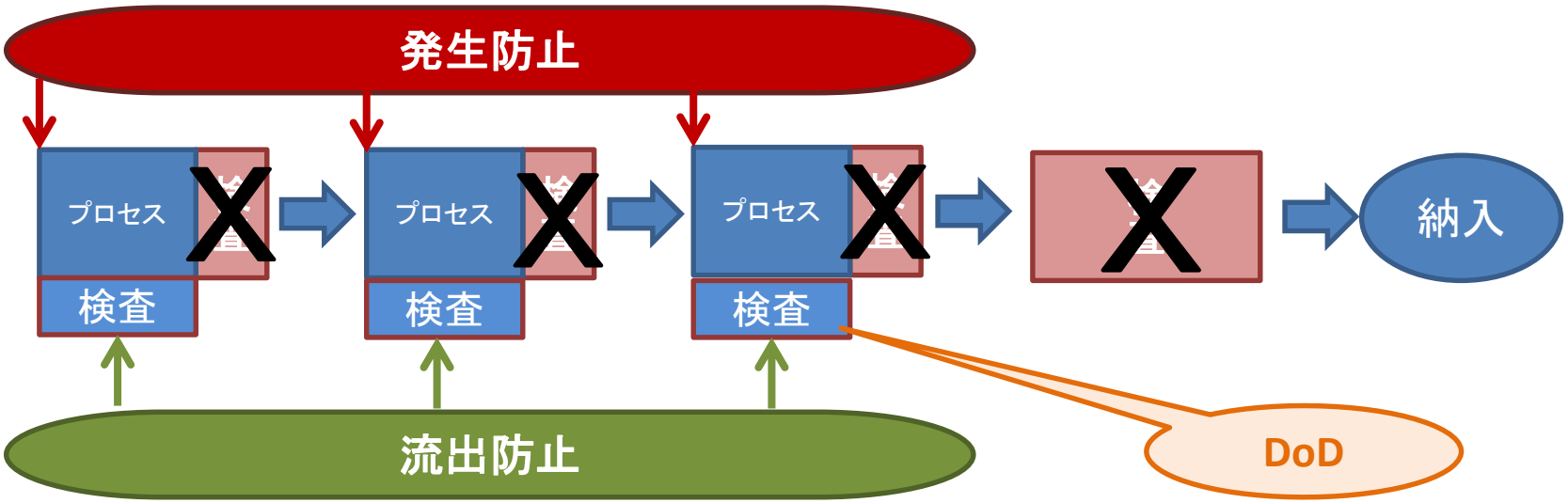
- オープンな作業環境
  - チーム全員のコミュニケーションがとり易く、一緒にいる時間が多くなるような環境が良い
  - 広い区画の無い共有作業スペース
    - パーティションは人を引き離し、チームを分離する

ペアプログラミングの風景



DoD (Definition of Done)　完了の定義

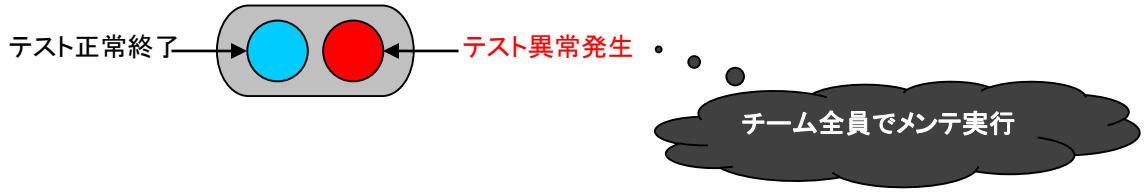
- 各作業の完了基準
- 閾値（標準値）を決める。
- 作業経過、結果を計測する。
- 自工程完結の基本的な姿勢（現場での意思決定）
- 仁＝他人の努力を無駄にしない思いやり



継続的インテグレーション（常時結合）

毎日実施

- 行灯（パトライト）
- 自動テスト



- 開発したプログラムを毎日統合して動作確認を行う
- システムの進捗を日々の動作機能で測る事が可能
- 開発チーム内での統合動作不良・改善を日々行う事が可能
- 自動化ツールを使用して開発チームが休息している間も統合試験が可能
- 品質向上に貢献

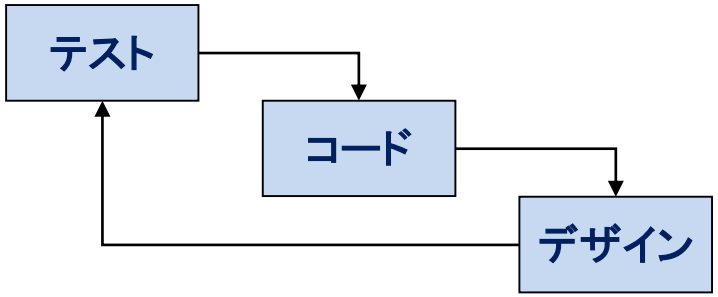
10分間ビルド

XP（エクストリーム・プログラミング）

- Kent Beck（1999年）
- テスト駆動開発（TDD）＝テストファースト・プログラミング
  ① 実装する機能をテストするプログラムを書く。
  ② コードを書いてテストする。
  ③ デザインを見直す。
  ④ 信号が青になるまで２＆3を繰り返す。
- リファクタリング
  - 完成したコードの見直し（実装された機能を変えずに、コードをシンプルに、見やすくする）
  - 任意の作業（全員が行う。 時間が空いたら行う。）
- ペア・プログラミング
  - ドライバー（コードを書く人）
  - ナビゲーター（コードをチェックする人、ナビゲーションをする人）
  - この役割を1日の中でペアの間で、途中で交代する。
  - ペアの組み合わせを毎日替える。
- １０分間ビルド
  - 自動的にシステム全体をビルドして、全てのテストを10分以内に実行させる。（理想形）

テスト駆動開発（TDD）

- **テスト駆動開発**
  - テストの自動化
  - アクセプタンス・テスト（完了・終了の定義）



小さいテストのコード作成
次にソース・コードを作成
最後にデザインをリファクタリングする。
このプロセスを廻す。

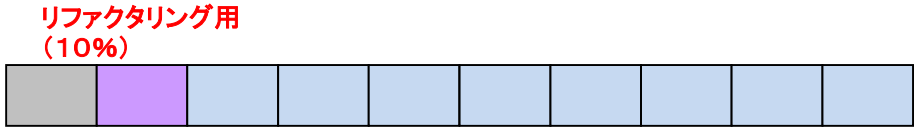
リファクタリング（Refactoring）

正常に動作確認したプログラムに対して下記目的でソースコードの調整を行う作業

- プログラムの可読性を高める
- クラス化・部品化
- 変更容易

仕様レベルでの調整

- 使い易さ
- 誤操作抑止
- 仕様変更



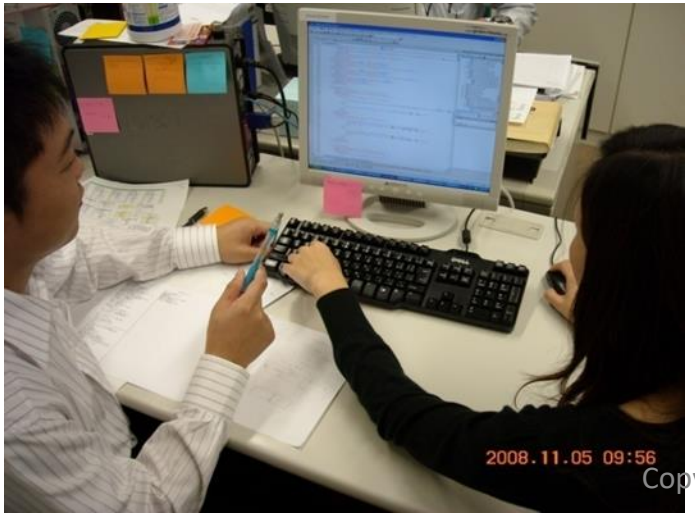
リファクタリングのルール

- リファクタリングと機能改変を同時にはおこなわない。リファクタリングをしてから新しい機能を追加する。
- リファクタリングを始める前と後にはユニットテストを実行しコードの機能が変更ないかを確認する。
- パフォーマンスよりもメンテナンス性を重視する。
- こだわり過ぎてはいけない。
- 小さなリファクタリングとテストの組み合わせを繰り返す。決して一度に大きなリファクタリングをしない。
- *大きなリファクタリングは惨事のもとである。*
  ～Kent Beck

ペア・プログラミング（Pair programming）

- 二人で一台の端末に向かってプログラム製造を実施する
  - 毎日違ったペア
    - 緊張感の維持
    - この瞬間にこの作業しか出来ない！！
  - 役割を一時間前後で交代
    - 作業者とチェッカー
    - 曖昧な作業が無くなる
    - 手戻りの軽減
  - 品質の向上
    - 動けば良いと言うような安易なプログラム製造ではない
    - 可読性の高いコード
  - 頭脳労働のムダとり
    - 一人で考え込む時間が無くなる。

ペアの交替（定期的：毎日）
作業の交替（一定時間毎）：ドライバーとナビゲーター

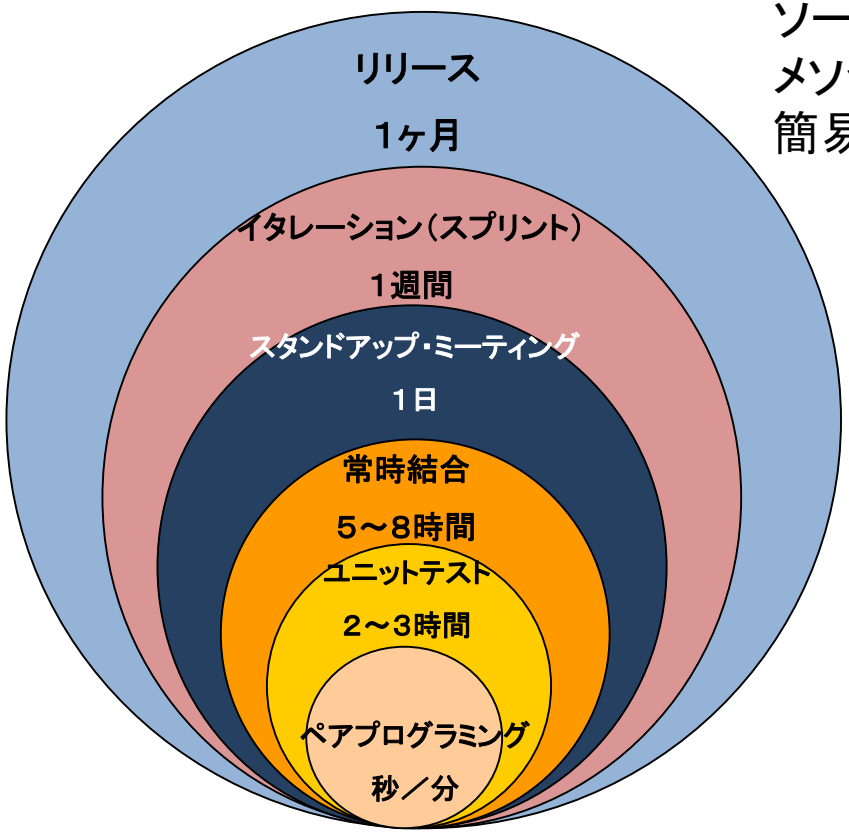


静かなペアプロは
機能してない

品質について考える

当り前品質 ⇒ 顕在品質 ⇒ バグフリー
アジャイル・プラクティスの徹底（守）
頻繁なフィードバック・ループの設計

魅力的品質 ⇒ 潜在品質 ⇒ メンテの容易性
ソースコード ＝ ドキュメント
メソッド名と変数名 （名は体を表す）
簡易に書けるセカンド・ランゲージ
（自分のツール作成）



アジャイル開発で品質が向上する理由

- ムダ取りの原則
  - 作業にムラがあるから、ムリをするようになる。
  - ムリな作業をした結果、ムダが生じる。
  - ムラを防止するのは、一定の作業リズム（タクトタイム＝タイムボックス）

- タスクの粒度を小さくする。
  - 作業手順（工程設計）を考えなければタスクは小さくできない。
  - タスクが小さくなれば、ミスを容易に発見できる。手戻りも小さい。
  - タスクを小さくするとムダが見えてくる。
  - 正味（本来の価値を作り込む）作業、付帯（事前、事後の）作業、ムダな作業
  - タスクを小さくすることで、整流化が容易になる。（ボトルネックの解消：　一個流し生産）

- 全員での作業で透明性が高まる。
  - 一人で抱え込む仕事がなくなる。
  - 事前に他人の目が届く（チェック）

- トヨタ生産方式（TPS）の自働化の思想をプロセスに組み込める。
  - 不良作業をしない。不良品を流さない。不良品を受け取らない。（自工程完結）
  - 不良が起こったらラインストップをして、関係者全員が異常に気づく。（見える化）

- 作業者（開発者）に直接フィードバックする仕組みが構築できる。
  - 『擦り合わせ』をしながら作業が進む。

タスクの粒度

- タスクの粒度を小さくすることはTPSにおける小ロット化と同様
  - 「流れ」を作り、負荷を平準化し、柔軟性を高める

Application size, Test Cases, and Test Coverage.
Logical source code statements
By Caper Jones

Statements of Source code	Test Cases	Test Coverage
1	1	100.00%
10	2	100.00%
100	5	95.00%
1,000	15	75.00%
10,000	250	50.00%
100,000	4,000	35.00%
1,000,000	50,000	25.00%
10,000,000	350,000	15.00%

- タスクの粒度は小さいほど良い
  - １日以内、理想は1時間
  - 責任を持って見積ができる
  - バグを作り込まない（簡単にテスト可能）
  - 他のペアと同期がとれる
  - ダイナミックなプロジェクト運営が可能となる（チーム編成の増減、分散開発など）
  - タスクが小さくできないのは、作業対象の内容把握に問題が存在するのではないか？
  - タスクを小さく分割するという事は、作業指示書を作成する事。

タスクを小さく（粒度）する

例えば、レポートを作る業務（仕事）をタスク分解する。

1. レポートの主旨を確認し、レポートのストーリーを練る。	30分
2. レポートの章立てを決める	10分
3. 各章の基本を決める（文章、図、グラフ、データ、イラスト等）	30分
4. 文章の下書きをする。	30分
5. 図やグラフを作成するためのデータを決め、データを収集する。	30分
6. PCを立ち上げ、EXCEL、PowerPointを起動する。	5分
7. データをEXCELに入力する。（データをインポートする。）	20分
8. グラフを作成する。	10分
9. 文章を構成し、PowerPointに入力する。（コピーする。）	30分
10. PowerPointのレイアウトを決め、文章、図、グラフ、イラストを配置する。	5分
11. ④～⑩を必要ページ分繰り返す。	
12. レポート全体を通して確認する（校正する。）	30分
13. 作成日、作成者名、レポートの題名を記入し、完成させる。	5分
14. レポートを提出する。	5分
	240分
	（4時間）

テーマ	いつでも安全に簡単に買い物ができる									
エピック（フィーチャー）	Ⅰ	買い物ができる	Ⅱ	売上管理	Ⅲ	商品管理	Ⅳ	顧客管理	Ⅴ	全体（共通）
ストーリー	1	顧客として欲しい商品を選択したい								55.5 時間
タスク	1-1	DB 構築								1
	1-2	商品テーブルの作成								0.5
	1-3	商品データを作成する（本）（100件）								3
	1-4	商品データを作成する（日用雑貨）（200件）								6
	1-5	商品データを登録する（写真以外）								0.5
	1-6	商品写真を用意する（本）（100冊）								2
	1-7	商品写真を用意する（雑貨）（200件）								4
	1-8	商品データを登録する（写真）								6
	1-9	商品一覧画面の作成（グルーピングなし）								1
	1-10	商品一覧の表示（DBから取得）（グルーピングなし）（商品コード順）								2
	1-11	商品一覧をグルーピングして表示する（タブ表示）								1.5
	1-12	CI（常時結合）環境の構築								6
	1-13	商品一覧テスト作成								1
	1-14	商品一覧検索（検索条件①商品名②価格③商品種別）								3
	1-15	商品一覧検索テスト作成								2
	1-16	商品一覧ページング								2
	1-17	商品一覧ページングテスト作成								1
	1-18	商品詳細画面の作成								2
	1-19	商品詳細画面に商品情報を表示する								1
	1-20	商品詳細画面から商品一覧画面に戻る								0.5
	1-21	商品詳細画面テスト作成								1
	1-22	買物かご画面の作成								1
	1-23	商品選択（商品詳細画面から買物かごへ）								0.5
	1-24	商品一覧画面から商品を選択し買物かごへ（複数選択する）								0.5
	1-25	買物かご確認画面の表示								1
	1-26	買物かご確認画面から一覧画面に戻る								1
	1-27	買物かご確認画面から商品詳細画面に戻る								0.5
	1-28	買物かご確認画面表示のテストを作成する								2
	1-29	買物かごから商品を削除する（買物かご画面に戻る）								0.5
	1-30	買物かごに入っている商品の数量を変更する								0.5
	1-31	買物かごでの変更処理（削除、数量変更）のテストを作成する								1

経験事例の報告（エンジニアの声）

エンジニアAさん： アジャイル初体験、ウォーターフォール開発経験約１５年（主任クラス）３７才
＞情報処理技術者　１種　＞UMTP　L1
＞業務SE（物流／生産管理／公共サービス）　８年　＞ホスト汎用機テクニカルSE　７年
＞今回　．NETは初めての経験　＞JAVA（web）　３年
＞PL／SQL　２年　＞VB．NET　（２週間：本アジャイル開発で初）
＞COBOL、C＋＋　etc（細かい開発は多数）

アジャイル開発の効果： コーディングの生産性は変わらないが、開発作業内で手戻りが無く、結果的に早く完了できる。

体験した感想： とにかく頭が疲れる。集中する。

・品質が高くなる。
・技術的な問題や、方式で悩む時間が少ないので効率は良い。
・気を抜く暇がないので、稼働率は高い
・二人でやっているので、生産性は倍まではいかない。ただし品質が高いので、改修やテスト時の修正工数は少なくなる
・ペアプロ／クロスファンクションにより　ソースコードレベルで情報を共有するため、自然に　可読性／ロジックのシンプルさが感じられる実装となる。
・随時に動かしながら機能拡張をするため、潜在バグ／デグレードのリスクは低い。
・実装が不慣れな要員がいても、ペアの組み合わせにより品質の高い実装が可能となる。
・作業の完了が、視覚的に理解できる。実装の成果がすぐに見れる。
・スタンドアップミーティング／振り返り／タスクの割り振りによりメンバー全員が全体の作業を見渡せる。司会を持ち回りすることにより参加意識が強調される。
人に見られているのに、適当な（動けばいい）コーディングはできない。
・悩んでいる時間が少ない。（随時相談／調査）
・具体的な目標を随時持つことができる。
・タスク担当を明確にすることにより、責任範囲の当事者意識を持つことができる。

経験事例の報告（エンジニアの声）

エンジニアBさん：　アジャイル初体験、ウォーターフォール開発経験約5年　28才
　　＞ Oracle Master Bronze 10g　　＞ VB.NET：1年
　　＞ HTML、JavaScript、CSSなどWeb関係：1年（随時）　　＞ PHP：1年
　　＞ VB6：3年　　＞ PL/SQL：1年
　　＞ XML：1ヶ月

アジャイル開発の効果：　生産性が上がった（体感）。

体験した感想：　個人のコミュニケーション能力が高く問われる
・二人で作業を行っているため、単純ミスも少なく精度が高い。
・思った以上に疲れる。気を抜く暇がない。

エンジニアCさん：　アジャイル初体験、ウォーターフォール開発経験約4年27才
　　＞ 初級アドミニストレータ
　　＞ 詳細設計・PG・/テスト（物流／在庫管理／Web）　4年
　　＞VB．2年　　＞ JAVA（web）　3年
　　＞ PL／SQL　三ヶ月

アジャイル感想：
・一人で開発をしている時は、技術的にわからないことがあるとネットや本等で調べて解決し、作業を進めていくが、ペア・プロの場合横で他の人が見ているので、気軽に調べ物がしにくい。
（まだメンバーに慣れていないため、気軽に質問ができない）
・仕様をよく知っている人とペアを組むと、プログラミングの道筋を立てて開発ができるので、良いと思う。
・反対にある程度わかっている人が作業を行い、ほとんどわかっていない人が横で見ている場合、作業者はどこまで説明しながら進めればいいのかわからない。
・ペア・プロだと常に他の人と一緒に開発を行うので、緊張感が保てる。その反面、気を抜くことができないので、一人で作業をするよりもかなり疲労度が高い。
・タスク毎に見積もり時間を出して作業を行うことと、プログラム1本の見積もり時間しか出ていない状態で作業を行うことと比べると、タスク毎の方が作業を進めやすい。
（プログラム1本の見積もりの場合、ペース配分が上手くいかないことがある）
・ウォーターフォール型の開発に慣れているので、要件定義からいきなりプログラミングに入ることにまだ戸惑いを感じる。外部設計書、内部設計書があった方が開発がしやすい。
・ペア・プロに慣れてきたので、作業進捗が早くなってきた。
・常に隣に他人がいる環境でずっと開発を行うのは疲労度が高く、辛い時もある。

日経コンピュータ連載コラム

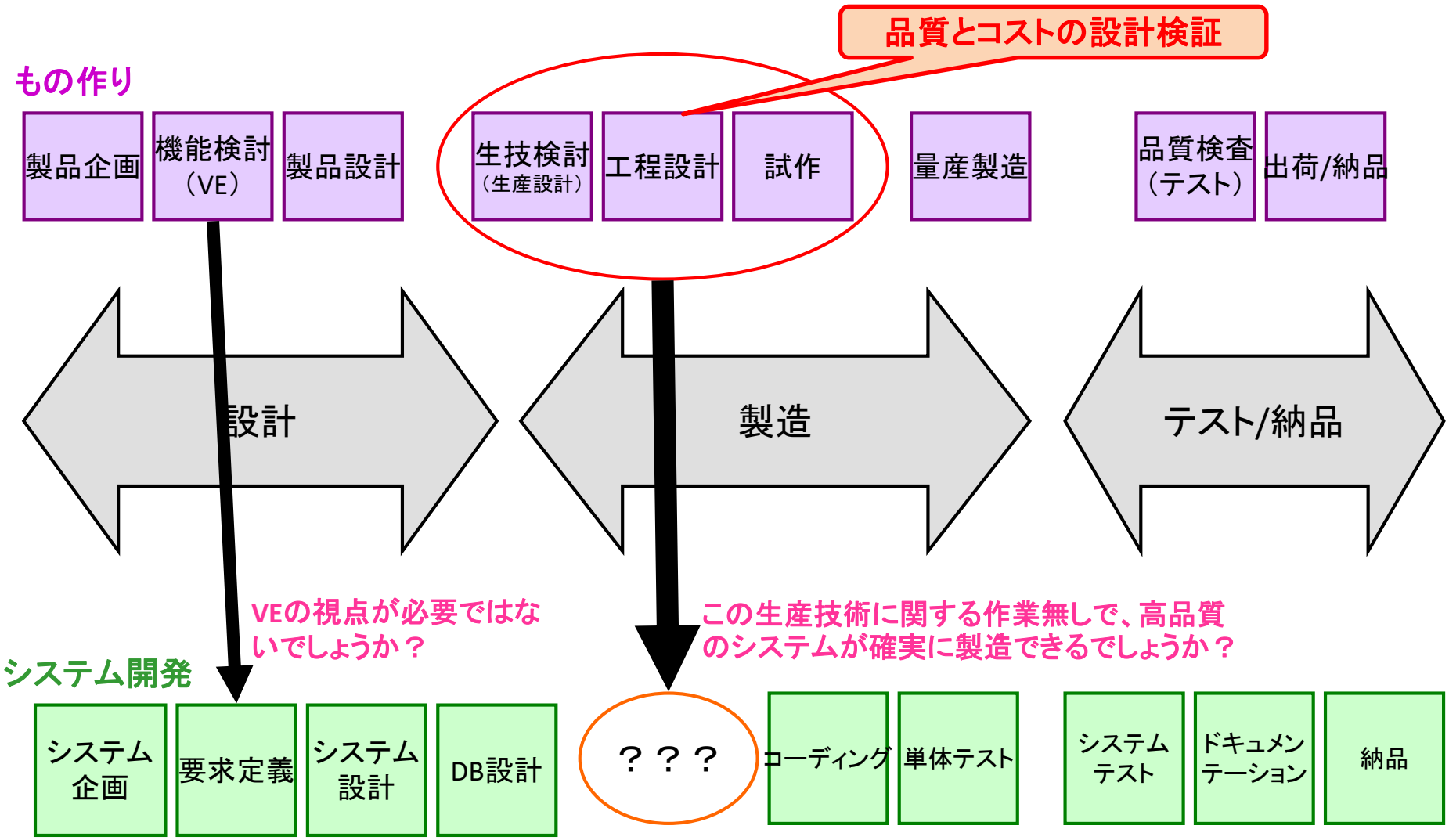
総合タイトル：前編 【現場を元気にするチーム運営術】

1. 2017年4月13日　元気が無い現場に四つのサイン あいさつが元気を取り戻す第一歩
2. 2017年4月27日　コミュニケーションが取れない組織 他人をどれだけ知っているのか
3. 2017年5月11日　残業が減らない組織 属人的な仕事を無くしているか
4. 2017年5月25日　多忙で夏休みを取れない職場 改善活動の停滞乗り越え取得率向上
5. 2017年6月8日　助け合いの無いシステム部門「心のマネジメント」が改善のカギ
6. 2017年6月22日　アジャイル開発でチームを元気に 初挑戦でも成功、カギは見える化
7. 2017年7月6日　発注者の信頼をどう得るのか アジャイル開発成功のカギ
8. 2017年7月20日　チームの透明性はアナログで高める　自らを律するのが成功の秘訣
9. 2017年8月3日　暗すぎるウォーターフォールの現場　やる気引き出す心のマネジメントを
10. 2017年8月17日　進捗が90%から先に進まない　1時間の作業に分割しているか

総合タイトル：後編 【現場を元気にするDevOps 2.0】

1. 2017年9月14日　今こそ大量生産時代の常識に決別　トヨタ由来の方法論で元気になろう
2. 2017年9月28日　自ら改善を続けるチームに変える　働き方改革への第一歩
3. 2017年10月12日 人を大切にするのが「強い企業」 仕事のやり方を変えて近づこう
4. 2017年10月26日「サイロ」現場を3段階で変革　個別最適では幸せにはなれない
5. 2017年11月9日　アジャイル開発の品質向上策　7つの視点で作業の様子を観察
6. 2017年11月23日「伝達ゲーム」は開発失敗の元凶　業務プロセスの観点を貫こう

『もの作り』と『システム作り』の相違



設計仕様を図面上のみの検討で、十分でしょうか？

擦り合わせ、微調整が必要ではありませんか？（誰が、何時、どの様に作業できますか？）