

第20期 ITソリューション塾

# アジャイル開発とSIビジネス

2015年11月25日

戸田 孝一郎

# 自己紹介

1971年4月 日本アイ・ビーエム株式会社 入社

鉄鋼業、製造業の生産管理システム、設計開発(CAD)システムを主に担当

1998年9月 株式会社アマダ 入社

米国子会社アマダアメリカ副社長兼CIO兼構造改革プロジェクト・リーダー

- SAP(ERP)を6ヶ月で稼働(2001年当時の最短稼働記録)
  - 並行してCRM(シーベル)を4ヶ月で稼働
  - ネットワーク・セキュリティ強化のネットワーク再構築
  - Webマーケティング・システム構築、
  - B to Bオンライン・ショッピングサイト構築
- SAPプロジェクトの開始からショッピングサイト構築までの上記システムを1年以内で立ち上げ  
全てのITプロジェクトは、6ヶ月以内で完了 (Jack Welchに学ぶ)

2003年10月 ペンタックス株式会社 入社

北米センター長 兼 米国子会社社長 兼 CIO

- SAP(ERP)を12ヶ月で稼働
- カメラ事業グローバル・ロジスティックス・システムの構築
- グローバル連結決算システムを3ヶ月で構築

2006年11月 株式会社戦略スタッフ・サービス設立

- アジャイル開発チームの育成とプロジェクト指導(20プロジェクト、300人)
- TPS/TMSでのプロセス改善の指導(6改善塾)

2009年 日本IBM(株)認定インストラクター

2008年 認定スクラムマスター(米国スクラムアライアンス)

2014年 TMS講師(TPS検定協会)

2015年 EXINアジャイル・スクラム・ファンデーション認定講師(EXIN)

EXIN Japan テクニカル・アドバイザー



◆ ビジネス戦略、IT戦略

◆ ビジネスモデルリング

◆ リーンスタートアップ

◆ TPS/TMS改善指導(自律した人財育成)

◆ アジャイル開発(人財育成&プロジェクト管理)

# フレデリック・テイラー(テイラー主義)

Industrial Engineering(IE)の始まり

工場の生産性を体系的に向上させる『科学的管理法』

工場の生産性を高める為に科学的手法(観察、仮説、実験)を適用して、作業者を観察し、最も上手くゆく方法(唯一最善の方法)を選択して、後は生産性の向上を保証するために工場全体で作業を標準化する。

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。

その結果、

- ◆ 計画立案と実行作業の分離
- ◆ 独立した品質管理部門の設置

# トヨタ生産方式(TPS)

19世紀からの産業界の常識を覆す。


昭和20年8月15日豊田喜一郎社長から大野耐一氏へ『3年でアメリカに追いつけ』

日本とドイツ1対3、ドイツとアメリカ1対3＝日本とアメリカ1対9(10倍の生産性)

- ◆ 顧客に価値を提供する事が最優先され、それを阻害するモノはムダ。
- ◆ 局所最適よりも全体最適を優先する。
- ◆ 仕事をまとめない方が効率が良い。(1個流し)バッチ処理の排除
- ◆ 計画は常に変更する。(計画の中身よりも計画作り)
- ◆ 現場に知恵が存在する。(現地現物)したがって作業手順や標準は現場で作る。
- ◆ 業績は、人のやる気(モチベーション)で全てが決まる。(やらされ感の排除)
- ◆ 仕事の完了は、全プロセスの完了。(顧客への価値の提供)
- ◆ 仕事には、顧客の価値に結びつく正味作業と正味作業を行う上で、必要な付帯作業がある。仕事の時間には、正味時間(正味作業)、付帯時間(付帯作業)、更にムダ(何ら価値を生まない、必要の無い時間)がある。
- ◆ 見える化は、作業者の説明責任。異常が見える事(誰でも解る)を『見える化』と言う。
- ◆ 品質の課題は、現場でしか対応できない。
- ◆ 全ての作業が正しく行われなければ、ムダが発生する。(自工程完結)
- ◆ 異常を検知したら全てのライン(プロセス)を停止せよ。皆で異常はその場で対応。

TPSの多くの側面は、ソフトウェア開発との強い類似性がある。(Kent Beck)

無駄は罪である。一度に一つの事をする。最初から正しくやる。働き過ぎが生む悪循環。流れを作る。(Jeff Sutherland)

A dramatic sunset over the ocean. The sky is filled with large, dark clouds, some of which are illuminated from below by the setting sun, creating a golden glow. Sunbeams (crepuscular rays) are visible, radiating from the horizon where the sun is partially obscured by clouds. The ocean is dark, and a small silhouette of a person stands on a rock in the foreground. The overall mood is serene and powerful.

DevOps = Development + Operation

# 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)中心で活用

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

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)

EXIN ITIL上級  
(MALC)

EXIN Agile Scrum  
Foundation (ASF)

EXIN ITIL関連  
(SOA,PPO,OSA,RCV)

バッチ・プロセス  
から  
一個流しプロセス  
へ

# ユーザーにITサービスをタイムリーに提供する手法

## DevOps



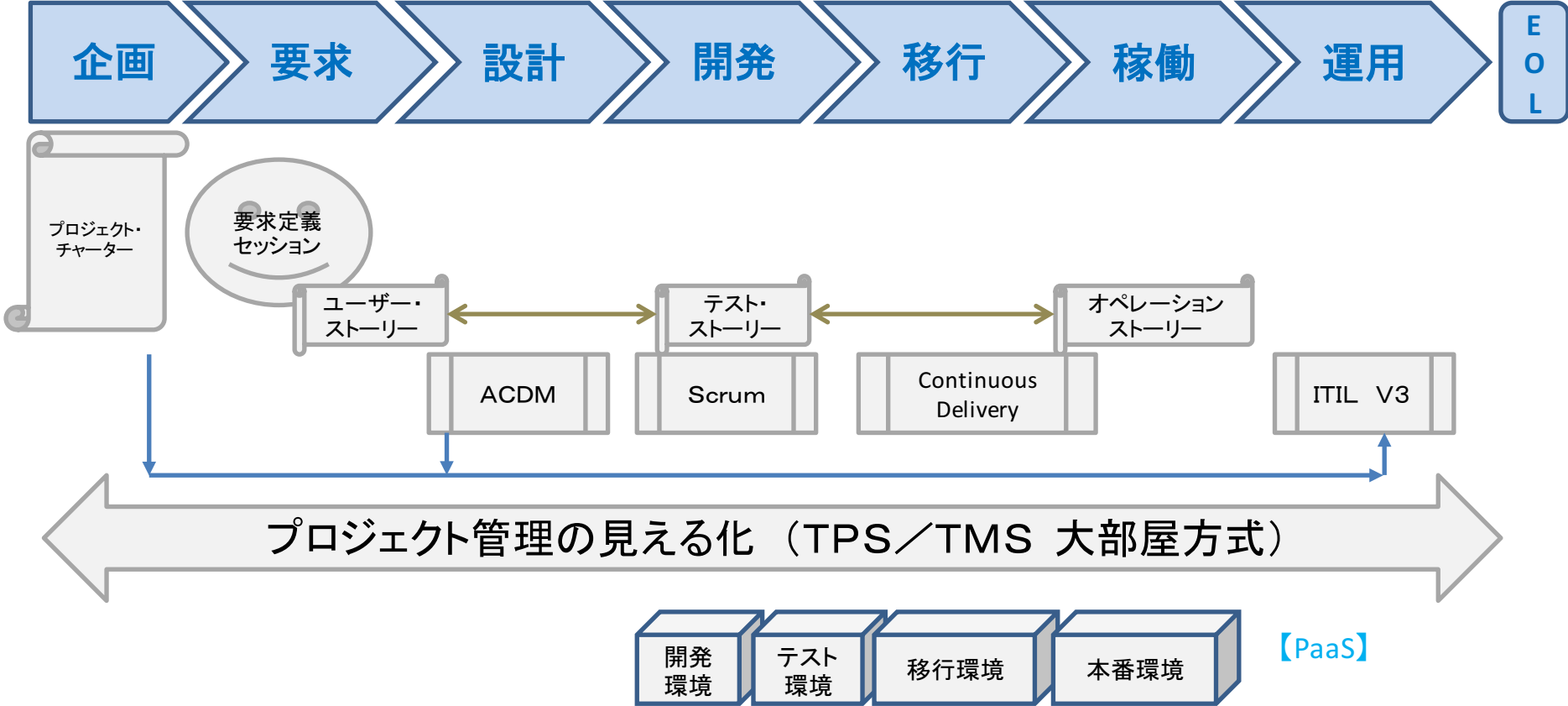
企画からEOL迄、一貫したプロセスの見える化と管理

【トヨタ生産方式(トータルTPS/TMS)と言われる大部屋方式】

ITサービスを提供する関連情報の一元化(共有化)も必要

特にBuild, Test, Deployの手順化&自動化の検討も必須

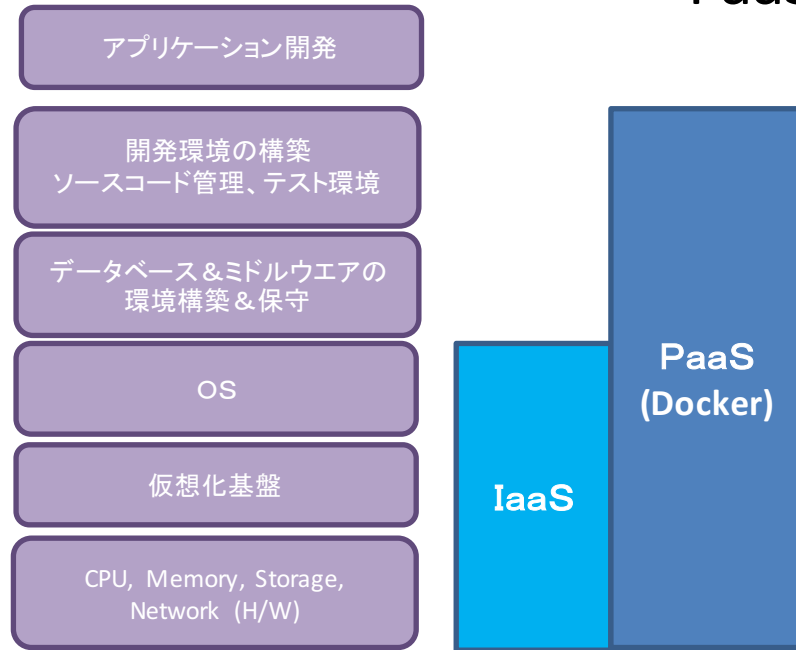
# Enterprise DevOps





# DevOpsのスピードと柔軟性を高めるインフラストラクチャー

## PaaS + Docker



吉田パクえさん資料「捕鯨！ 詳解docker」より抜粋  
[http://www.slideshare.net/yuya\\_lush/docker-42739730](http://www.slideshare.net/yuya_lush/docker-42739730)

アプリ開発者が求めるものは？

- ・完成したものをそのまま持ち込みたい (もしくは、全く同じことを保障してほしい)
- ・VMではサイズが大きいのので運びにくい
- ・作るのをもっと早くしたい

ここだけで何とかしたい!

昔からあった技術の復興 ~ コンテナ

コンテナ

Build, Ship and Run Any App, Anywhere

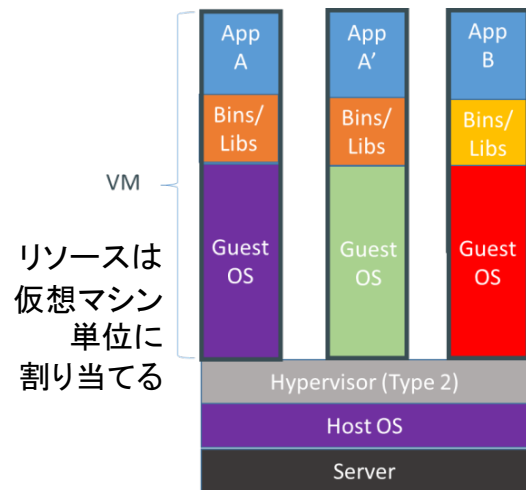
Docker - An open platform for distributed applications for developers and sysadmins.

ミドルとアプリで塊を作り、その下に基盤を設けることでどこでも動く状態を確保する

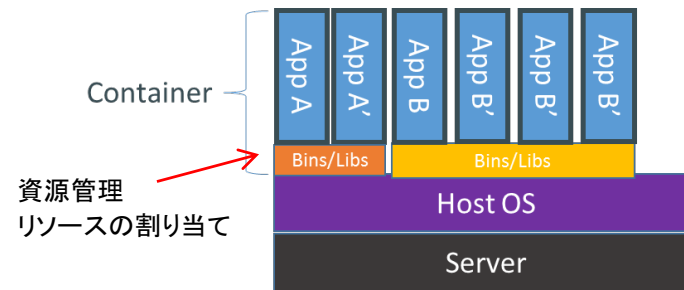
Docker Hubでの公開イメージを活用

無料イメージを使いコンテナを作成する。稼働させるソースコードは独自のものを準備する流れです。

最新版のRubyやRailsのインストールを割愛できる



仮想マシンによる仮想化



コンテナによる仮想化

# まとめ

ビジネスのスピードを牽引するジャストインタイムでのITサービスを提供する為には、、、

- ◆ 変化に対応する機敏なアジャイル開発(タスクの粒度)
- ◆ 安定したITサービスの運用を担保するITIL® V3(環境のパターン化)
- ◆ 柔軟かつ機敏な稼働環境を作るPaaS + Dockerのクラウド環境

以上の三点が必須の要件です。

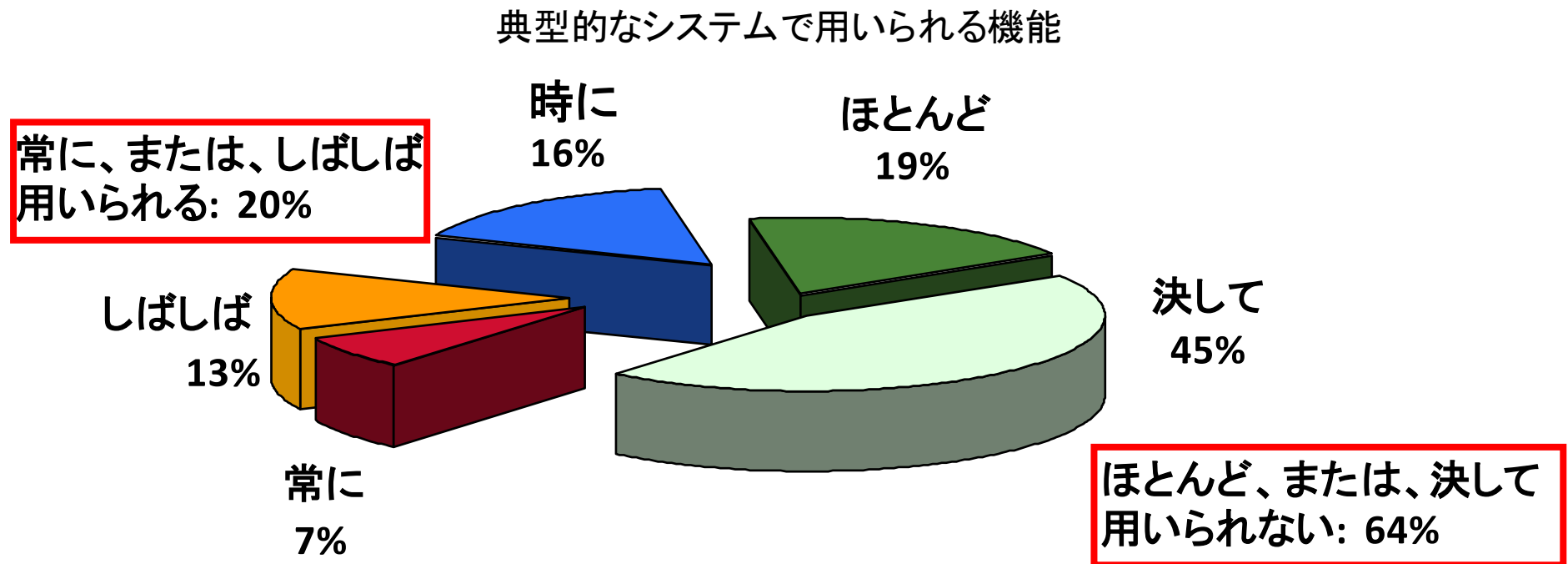
あとは、  
開発から運用までの全てのプロセスの整流化と全体最適を追求するカイゼン活動が必要です。

# アジャイル開発



# システム開発の不都合な真実－1

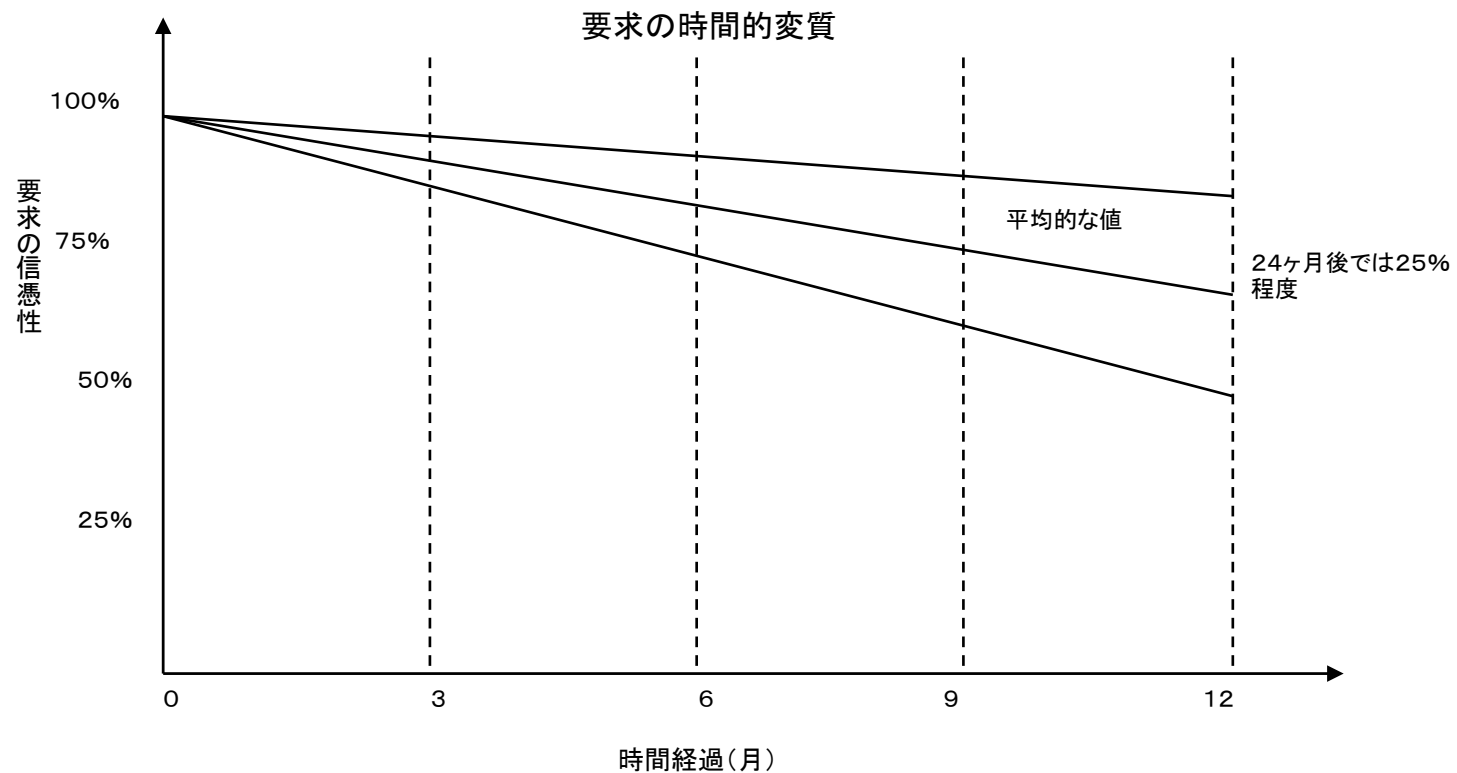
## 早期の仕様確定がムダを減らすというのは迷信



Standish Group Study Reported at XP2002 by Jim Johnson, Chairman

## システム開発の不都合な真実-2

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



## システム開発の不都合な真実－3

QAの体制、手法を強化すれば品質が上がるという迷信

1000行当たりのバグの発生率を管理する意味???

(統計的品質管理)

そもそもバグとは品質の問題???

(不良作業)

優秀なプロジェクト管理者を配置すればプロジェクトが  
上手く行くという迷信

コンテンジェンシーを見込めば、リスクが軽減するという迷信

PMBokに沿ってプロジェクトを推進すれば上手く行く  
という迷信

# アジャイル開発とは？？？

- プログラムの開発テクニック(手法)

  - プロジェクト管理手法

- システム開発ビジネス・プロセスの変更(変革)

  - 全てのビジネス・プロセスの変革&改善

# アジャイル開発ー概論

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ェ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)  
    アリストアー・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・ポップエンディーク(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

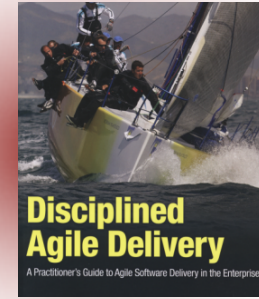
Dave Thomas

<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ェ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. 稼動するソフトウェ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. ソフトウェ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。

# アジャイル開発の発展



## 規律あるアジャイル開発



## スケーリングアジャイル開発



## 古典的アジャイル開発

XP DSDM  
スクラム FDD

## ビジネス価値とアジャイル開発の融合

リーン



2000年 2002年

2004年 2006年

2008年 2010年 2012年

# アジャイル神話 誤った理解

- 早くできる(作れる)

- 安く作れる(コストが下がる)

- 開発者が楽しく働ける

やはり、デスマーチ

納期も何時できるか？解らない

## アジャイル開発に対する疑念 & 誤った理解

- ✓ アジャイル開発に適したプロジェクトとは小さなWeb系のシステム???
- ✓ 全てを開発者に任せてプロジェクト管理が出来ず納期を守れないのでは???
- ✓ 変更を受け入れると、ユーザーからの変更要求が増加しエンドレスなプロジェクトになるのでは???
- ✓ 開発者(プログラマー)だけが楽しく仕事できる方法???
- ✓ 二人でプログラムを作成(ペアプロ)して本当に生産性が上がるのか???
- ✓ ユーザーとの協業なんて言っても、ユーザーが協力してくれる訳がない。
- ✓ インキュベーション(気の合った仲間)や小さな組織には向いているが、大企業では不向き???
- ✓ しっかりとしたアーキテクチャーやDBを設計できない。
- ✓ 品質管理が開発チーム任せで品質が保てない???
- ✓ エキスパートがやる開発手法では???
- ✓ ドキュメントを作らないって無管理状態???
- ✓ それでプロジェクトが上手く行く筈がない。
- ✓ PMBoKやCMMI, ISO(ガバナンス)の基準に合わないから使えない。
- ✓ プログラミングとは創造的な仕事だから、クリエイティブな作業ができる静かな環境(個人ブース)が必要。

# アジャイル開発の基本 見えるプロジェクト管理

水蒸気も集まって冷やされて雲になれば見える

開発現場では、色々な情報(データ)が既に存在している  
ただ、見ていない。観ようとしなない。

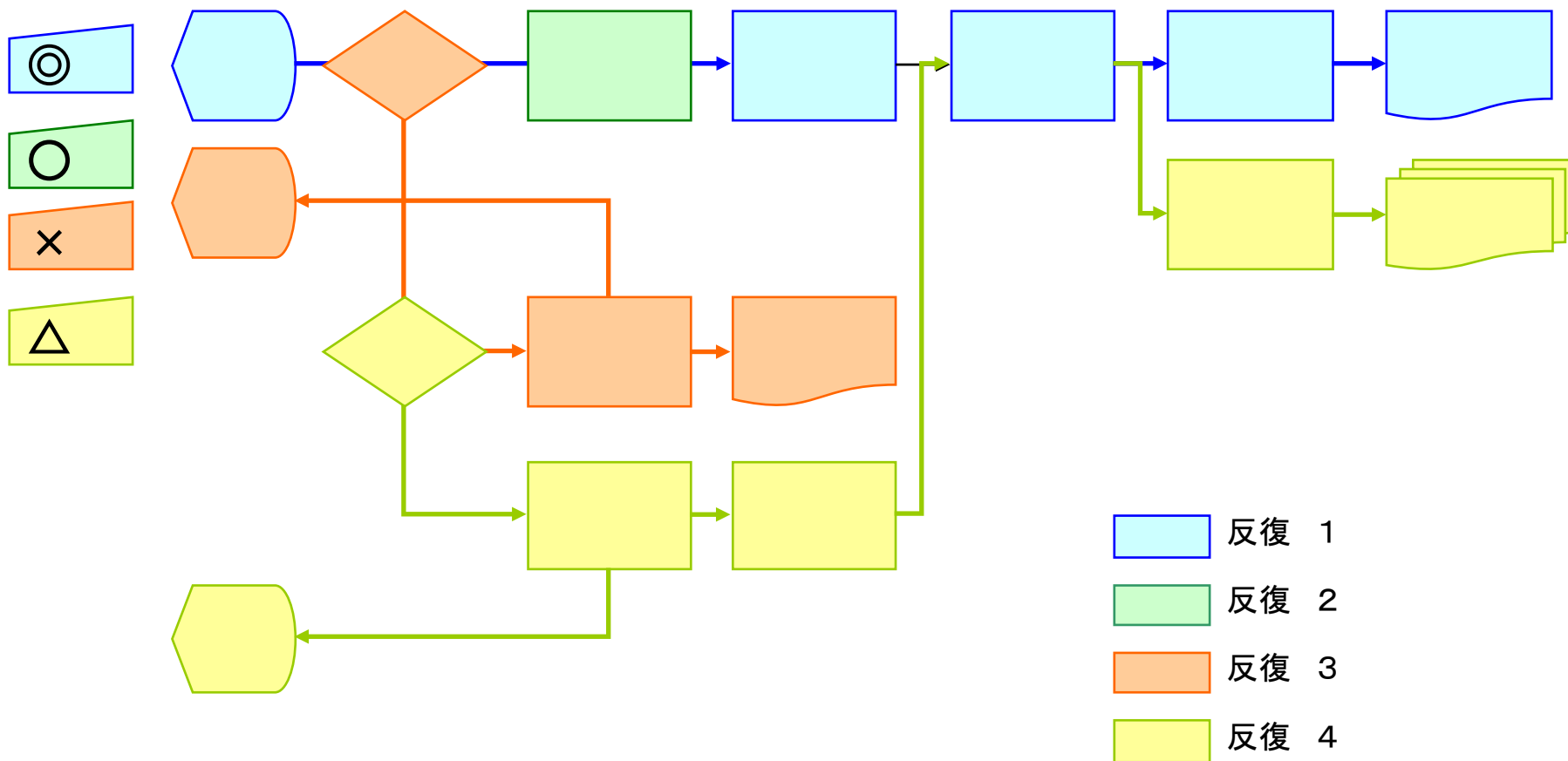
現場の説明責任

アジャイル開発の肝は、『見える化』と『カイゼン』

見えないモノを見る様にする

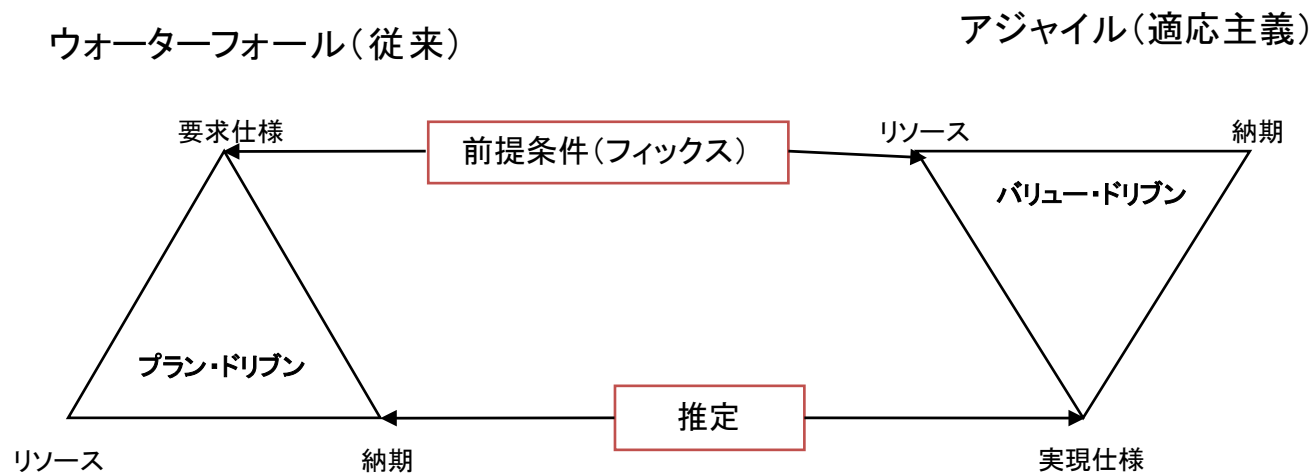
パラダイム・シフトが起こらなければ、見えない。

# アジャイル開発におけるソフトウェアの作り方



# アジャイル開発とはパラダイムシフト

## 価値観のパラダイムシフト





# アジャイル開発とはパラダイムシフト

- 大量生産（ベルトコンベヤーライン＋単能専任化）

人間の機械化（ロボット化）  
管理視点：統計的品質と稼働率  
プログラム化  
大規模で採算を取る

ウォーターフォール

- 一個流しの流れ生産（多能化＋作業責任、U字ライン&セル生産）

人間性の尊重  
管理視点：お客様満足と可働率（べきどうりつ）＝流れ（整流化）  
人間の思考力  
小規模でも採算が取れる

アジャイル

## 利用者(ユーザー)から見たアジャイル開発のメリット

顧客(ユーザー)から見てこのアジャイル開発手法はどのようなメリットを享受できるのでしょうか？もうシステム屋の言う事は信用しないという経営者の方も多いのではないのでしょうか？そうです従来の主流であるウォーターフォール型開発では米国の調査でも計画期間内、計画予算内でプロジェクトが完了した成功率は僅か16%です。

こうなりますとシステム開発プロジェクトは失敗して当たり前と言う評価を甘んじて受けざるを得ません。アジャイル開発では、このような失敗を回避可能です。と言いますのも、アジャイル開発とは現有リソース(資源:人、金、時間)を前提に計画を立て実行する手法だからです。

### ◆ プロジェクト進捗の見える化

完了した働くプログラム本数(実現した機能の数)で管理

### ◆ 製作している機能の見える化

ユーザー用語での機能定義(理解できる言葉での設計確認)

働くプログラムを稼働させて確認

### ◆ 絶え間ない擦り合わせ技術での 品質向上

品質とは、要求品質、設計品質、実装品質、検査(テスト)品質

### ◆ 優先、重要機能(システム)の早期実現

システムを構成する重要機能から優先して開発

### ◆ 開発プロジェクト期間中での柔軟な変更対応

プログラム製作期間(イタレーション)に入る前までは、変更自由

### ◆ ユーザーが参加すべき作業の見える化

# 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)
  - 昨日横断的な固定化されたチーム
  - 持続可能なペース

## (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。

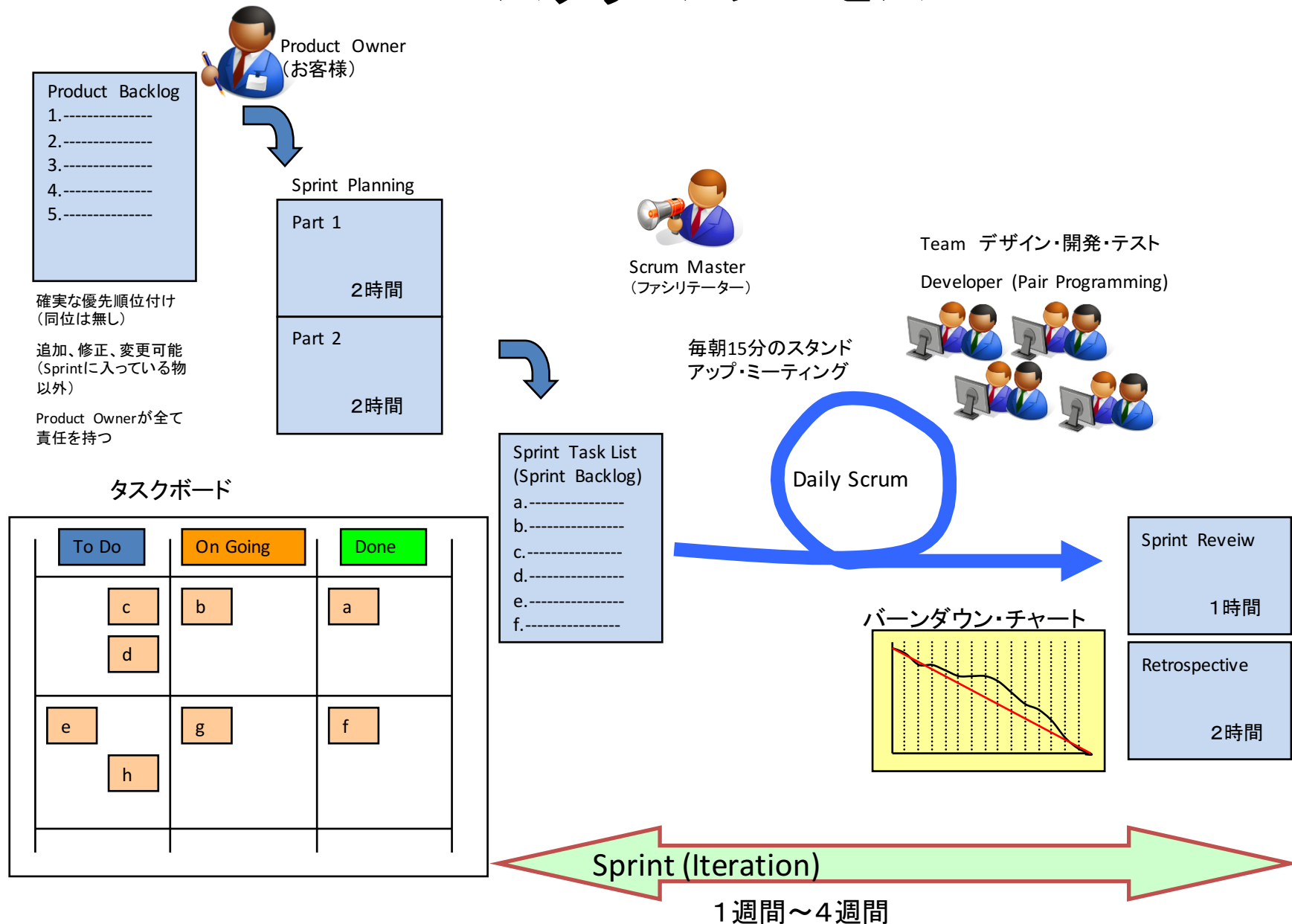
集中力は長く続かない。休む事で充電される。

時間が読めるからリラックスできる。

# スクラム 役割とプロセス

- 役割
  - 開発チーム
  - スクラム・マスター
  - プロダクト・オーナー
- プロセス
  - スプリント計画会議（パート1、パート2）
  - デイリー・スクラム（スタンドアップ・ミーティング）
  - スプリント・レビュー
  - スプリント・レトロスペクティブ（振り返り、KPT）

# スクラム・プロセス



## (参考) KPTの振り返り(レトロスペクティブ)

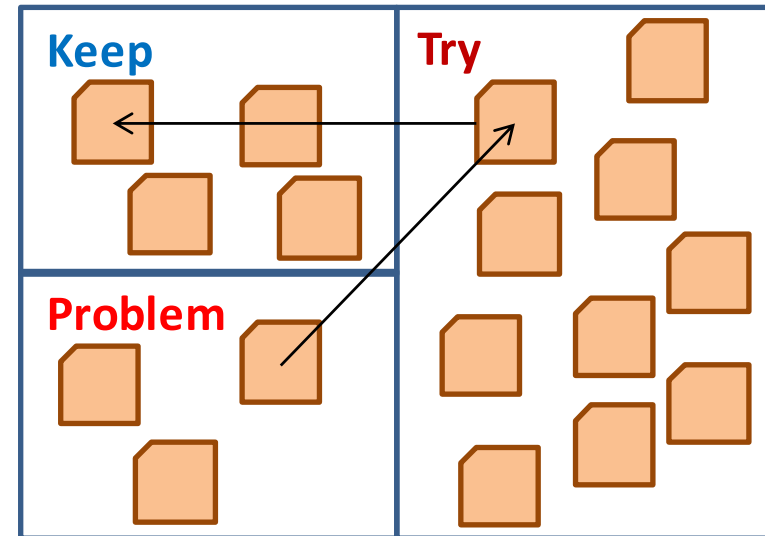
- 毎週行われる進捗報告ミーティング
- 評価とプロセスの改善の重要な機会
- PTKをチームが共有する
  - P(Problem: 問題となっていること)
  - T(Try: トライしたいこと)
  - K(keep: キープしたいこと)

小さなPDCAを回すための方法です。

KPTボードは振り返りのツール

- ①気づきや問題、課題を付箋紙に書き、Problem(問題)のエリアに貼る。
- ②Problemのエリアに出てきた気づきや問題に対してTry(対策)を立ててTryのエリアに移動させる。
- ③実際にやってみて、対策が不十分だった場合には、再度、Problem(問題)のエリアに移動し課題を追記する。再度、対策を立て直す。
- ④Problemに対する効果が十分出たと判断できた場合には、Keep(継続)のエリアに移動し習慣化、定着化を図る。

ここで大事な事は、人を責めずにやり方を責める事です。(誰でも同じようにできる様にする。)



# スクラム・プロジェクト

- 計画
  - 計画は常に作り直す。(『計画』よりも『計画作り』が重要)
  - 2レベル計画 (リリース計画、 スプリント計画)
  - 実測駆動の見積り
- タスク
  - WBSとは異なる。(作業指示書:手順を表記)
  - 粒度が重要(できるだけ小さくすると、平準化、多様化に対処できる。)
  - Done(作業完了)の定義
- 要求
  - ユーザストーリー
- 設計
  - プロセスで設計(ボディー・プロセス。 オルタナティブ・プロセス。 オプション・プロセス)
- コーディング
  - コードの共有(属人性、私物化の排除)
- テスト
  - 単体テスト(TDD)
  - 継続的インテグレーション[常時結合) = 行灯
- 管理
  - 目で見える管理
  - 働くコードで進捗
  - ベロシティーで納期(巡航速度)
- チーム運営
  - 自主研(自主改善活動) = 小さなPDCA
  - 報・連・相(毎朝のスタンドアップ・ミーティング)
  - モチベーション(ニコニコ・カレンダー)
  - 残業はチームで相談(全員で残業)



## 常時結合（継続的インテグレーション）

- 開発したプログラムを毎日統合して動作確認を行う
- 行灯（パトライト）が赤（テスト停止）になったら、全員で原因究明と回復作業を即実行する。
- 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

# 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)

# XP(Extreme Programming)

## プロジェクト管理の視点

Small Releases -- 短期リリース  
Whole Team? -- チーム全体  
Customer Tests -- ユーザーテスト  
Planning Game -- 計画ゲーム

## チーム運営の視点

Collective Ownership -- 共同所有  
Continuous Integration -- 常時結合  
Coding Standard -- コーディング規約  
Metaphor -- メタファ  
Sustainable Pace -- 最適ペース

## 開発の視点

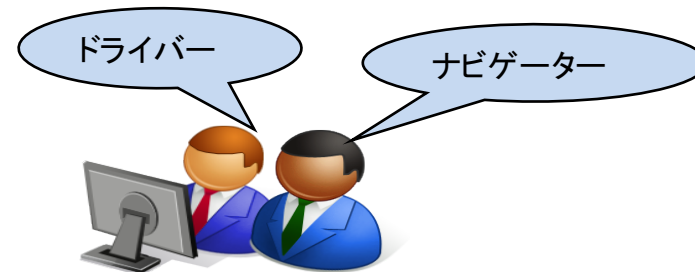
Simple Design -- シンプル設計  
Refactoring -- 設計改善(リファクタリング)  
Test-Driven Development -- テスト駆動開発  
Pair Programming -- ペアプログラミング

## 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

# ペア・プログラミング (Pair programming)

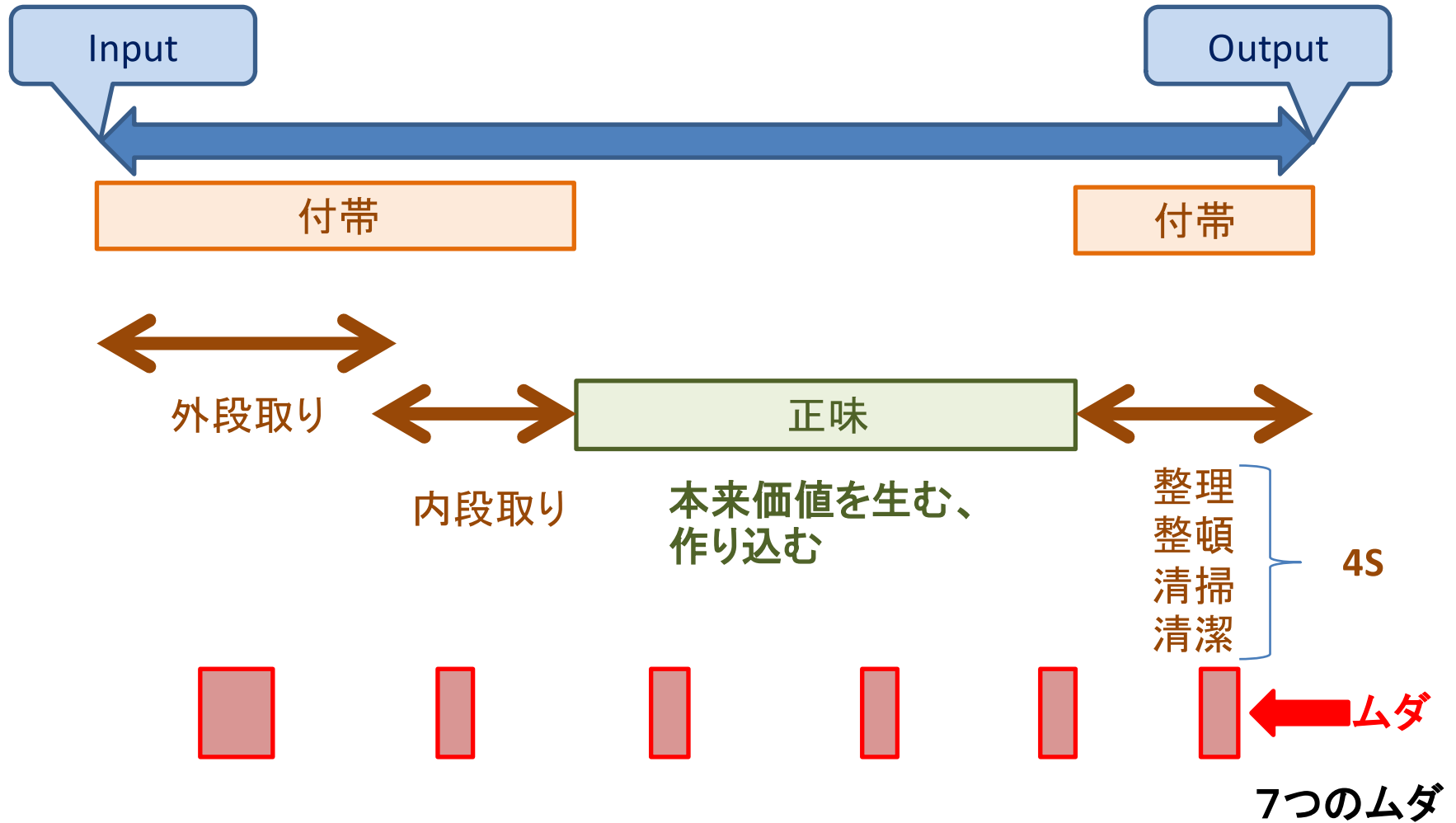
- 二人で一台の端末に向かってプログラム製造を実施する
  - 毎日違ったペア
    - 緊張感の維持
    - この瞬間にこの作業しか出来ない！！
  - 役割を一時間半前後で交代
    - 作業者とチェッカー
    - 曖昧な作業が無くなる
    - 出戻りの軽減
  - 品質の向上
    - 動けば良いと言うような安易なプログラム製造ではない
  - 頭脳労働のムダとり
    - 一人で考え込む時間が無くなる。



ペアの交替(定期的 : 毎日)

作業の交替(タスク、時間毎) : ドライバーとナビゲーター

# 仕事を分析する



- ①作りすぎ、②手待ち、③運搬、④加工そのモノ、⑤在庫、⑥動作、⑦不良を作る

# ソフトウェア製造工程におけるムダの廃除

## 1. 作りすぎのムダ

顧客に使用されない機能、実際には不要な機能、真のビジネス価値を生まない機能などの余分な機能を作らない。

StandishのCHAOSレポートによるとソフトウェアの全機能の64%は全くあるいは殆ど使用されていない。

## 2. 手待ち(停滞)のムダ

仕様の提示遅れによる製造開始遅れでの待機、許可待ち、ビルド待ち、障害発生によるテスト待ち

## 3. 運搬のムダ

複数プロジェクトでの作業の切り替え、仕様入荷チェック、納品出荷チェックの提供 & 受領双方での重複チェック

## 4. 加工そのもののムダ

開発現場で機能していない作業項目例えば余分な事務処理、報告書作成や作業分担の誤りによる過剰な作業

## 5. 在庫のムダ

最終工程で使用されない文書や計画、コンポーネントなどの中間的な作業成果物と待ち状態で仕掛中のプログラム

## 6. 動作(作業)のムダ

関係者の作業場所の移動や複数の開発ツールを使用して開発ツール間の切替・移行

## 7. 不良をつくるムダ

バグの作り込み---要求仕様(要件)、設計、コードの欠陥

# 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、ミスを容易に見見できる。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)の自動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)
  - 不良が起これば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(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<br>(4時間) |

# 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
- ＞ UMTF 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に、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を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。



# アジャイル開発の美味しい成果

事例1: アジャイル開発で受託した事例の紹介

事例2: ウォーターフォールで失敗したプロジェクトをスクラムでレスキューした事例の紹介

# アジャイル開発導入の効果

## 【生産性】

5年で1.7倍の生産性 (大手保険A社)

## 【品質の作り込み】

欠陥発生件数が5年で26%減少 (大手保険A社)

## 【進捗管理の透明性】

プロジェクト期間中での月次進捗会議の廃止 (中堅開発B社)

## 【ユーザーの信頼感】

プロジェクト期間途中での次期開発案件の委託要請  
(中堅開発B社)

# ケース・スタディー

## プロジェクト概要:

- 年商30億円(年率約10%成長)、JSDAQ上場企業(全国チェーン展開のホームリノベーション)
- 基幹業務システムの再構築(リプレイス)
- 約半年掛けて外部コンサルタント(ITIL)を交えてRFPを準備し、7社での公開入札コンペ  
3社が辞退し、4社での入札(内1社は現行システム構築)
- 新任のシステム管理部長が半年前に入社、(元大手インターネット証券企業システム管理者で前職はITコンサルタント) PMBOKに精通し、細かな指示、監督を行う

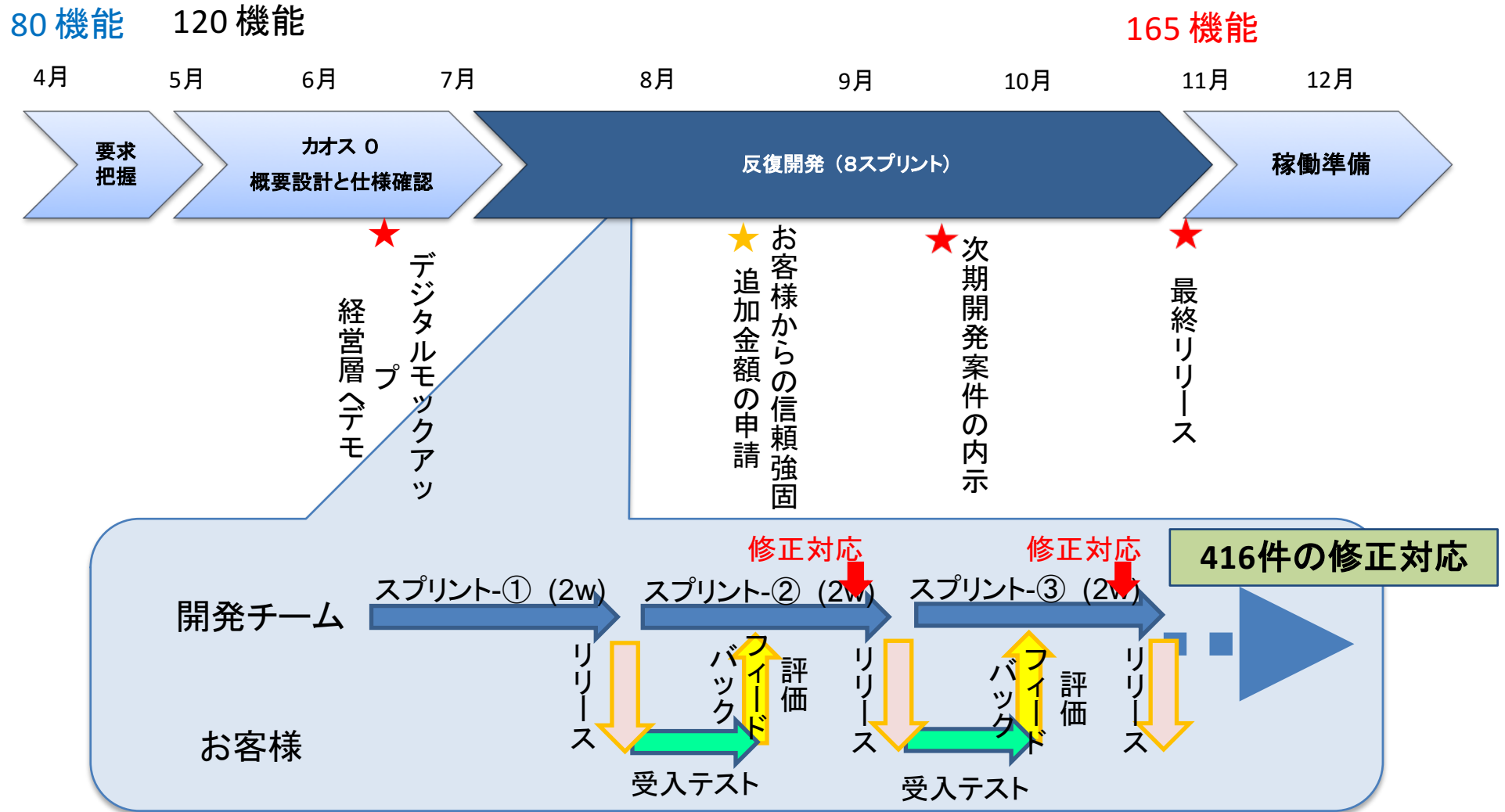
## プロジェクトの特長:

- ◆ ゼネコン一括受託方式に対抗して設計事務所+工務店方式での受託開発
- ◆ アジャイル開発を前面に出して提案(すりあわせソフトウェア開発の実践)
- ◆ リーンスタートアップ(E・リース)の実践

## 結果(評価):

- 予定通りでの新システムの稼働
- 非常に高いお客様の満足度(評価)で、
  - ①プロジェクト実施途中での予算の増額(+3,000万円)
  - ②プロジェクト実施途中に次期開発案件受託決定(約6,000万円)
  - ③プロジェクト実施途中に運用サービスの受託決定
- プロジェクト損益は営業利益ベースで30%~40%

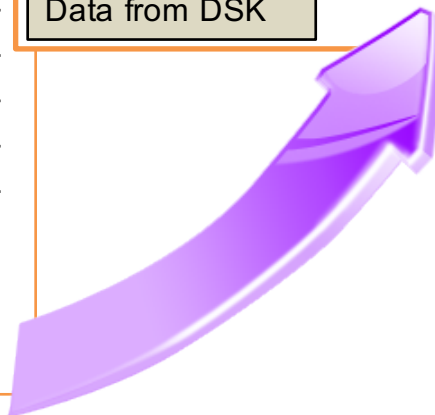
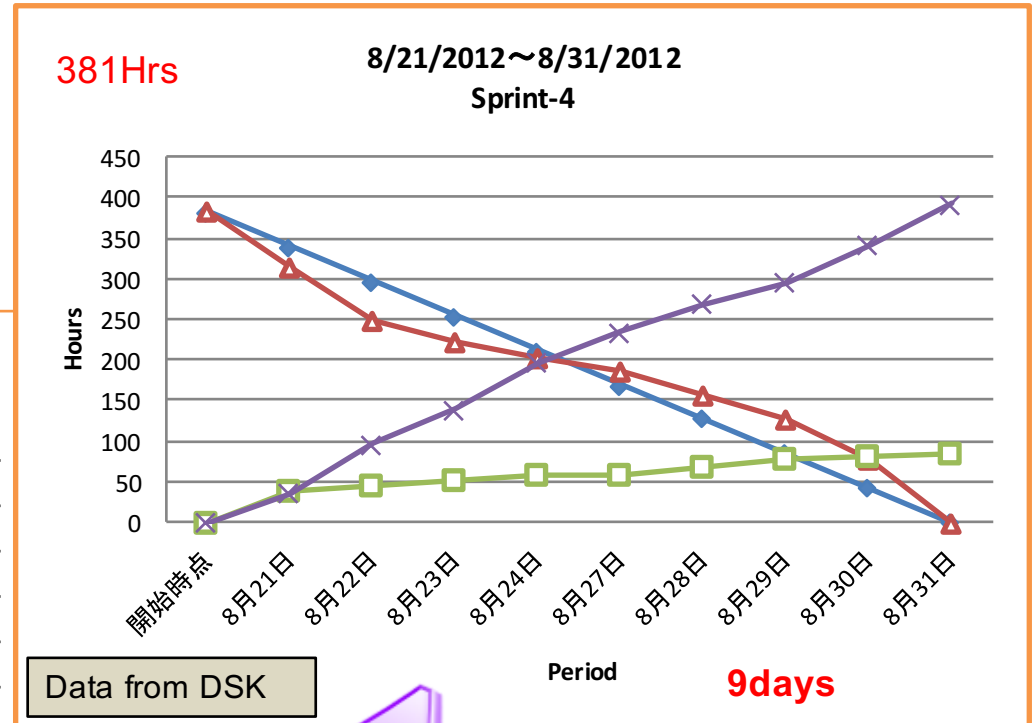
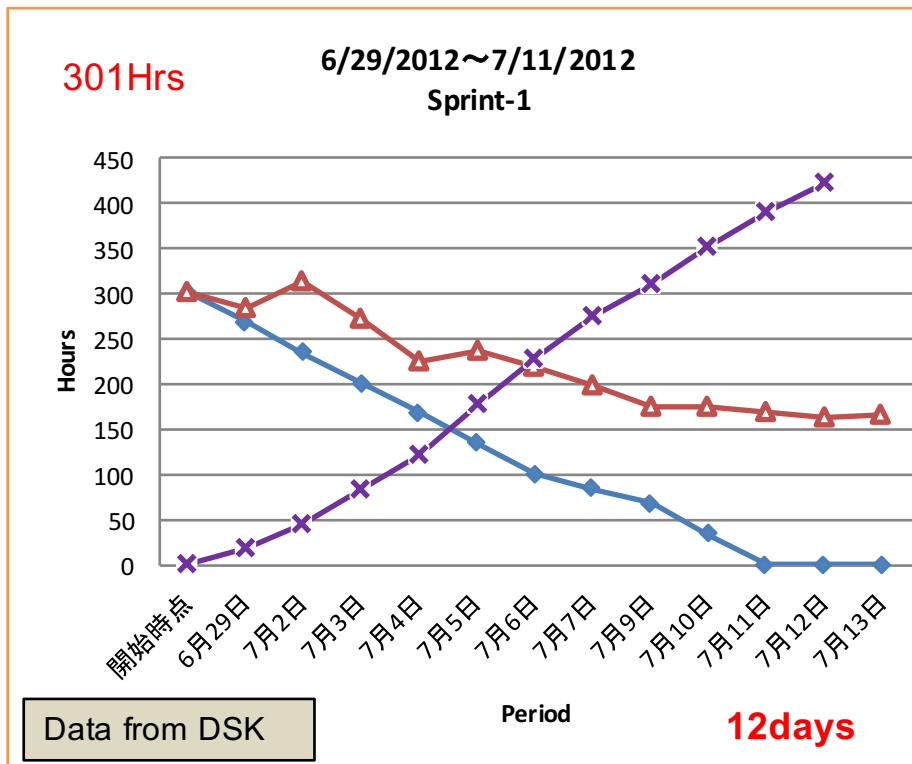
# プロジェクトの期間と運用



# チームの成長（カイゼン）

同一プロジェクト、同一のチーム、同一の手法（プラクティス）で実施しても  
何故、生産性が変わるのか？

- △ △ △ : Actual
- ◆ ◆ ◆ : Plan
- × × × : Actual (Accumulation)
- □ □ : Correspond to Feedback



生産性の向上  
6週間で168.7% 向上



# プロジェクト成功の理由

- RFP記載の280に上る実装機能要求を期間を限定して80機能という制限を提案し、要求の重要度&優先度、根拠、ビジネス価値、リスク、条件等の要求属性の明確化でリーンスタートアップのアプローチ
- 6ヶ月間で80機能の実装と言う開発チーム(10名)の明確なコミットメント
- カオス0(仕様確認期間)でのユーザー要求と実装方法の仮説検証をデモで確認(デジタル・モックアップ)
- プロジェクト期間を通して、開発チームの作業(進捗)の透明性を確保し、ユーザーからの強力な信頼感を獲得(チームの行動特性)
- 実装したコード(現物)を反復的(定期的)にユーザーにリリースし、『すりあわせ』による要求確認と変更への開発チームの素早い対処(修正実施)
- 徹底的にムダなドキュメンテーションを避けて開発(コーディング)作業に専念、しかしお客様のニーズ(要求)に応えたドキュメンテーション(データ)のJITでの提供
- 自律した開発チーム・オペレーション(毎週実施された現地現物での振り返り)

# FBI センティネル・プロジェクト（スクラムの事例）

2005年プロジェクト開始

予算4億5100万ドル、2009年完成予定

2010年3月時点

- 4億500万ドルを費やし、プロジェクトの進捗は50%（既に1年遅れ）
- 完成まであと6年から8年、追加予算見積もり3億5000万ドル
- FBIの契約スタッフ数220名、FBI職員数30名

スクラムでのレスキュー（案）

予算残額2000万ドルで12ヶ月以内で完成

FBIの契約スタッフ数40名、FBI職員数12名

結果

Sierが10年の歳月をかけ予算の90%を費やして完成出来なかったシステムを、  
20ヶ月の期間、当初予算の5%で、完成

# 輝く明日、アジャイル開発の発展

アジャイル開発の基本思想 = トヨタ生産方式(TPS)

アジャイル開発がちゃんとできれば、DevOpsも  
リーンスタートアップも確実に実現します。

# ご清聴ありがとうございました。

戸田 孝一郎

米国スクラムアライアンス認定スクラムマスター

IBM認定 アジャイル開発インストラクター

EXINアジャイル・スクラム・ファンデーション認定講師

(社)TPS検定協会 理事 認定TMSカイゼン塾 コーチ



<http://www.Ask3S.com>



株式会社 戦略スタッフ・サービス

(本社) 〒100-0004 東京都千代田区大手町1-7-2 東京サンケイビル27F 電話:03-3242-6282 FAX:03-3242-6283

お問い合わせはメール(lktoda@ask3s.net)にてお願いいたします。

# 講演・コラム 等の実績

- 2009年
  - SODEC 専門セミナー 講演
  - SODEC IBM社ブースにてアジャイル開発のセッションを担当
  - 日本IBM Rational Software Conference 2009 講演
  - 日経BP ソフトウェア技術者人材育成シンポジウム 講演
  - マイクロソフトのアジャイル開発支援サイトへの寄稿 (10回) (2009年～2010年)
  - マイクロソフト 最新.NETシステム開発の実践セミナー 講演
- 2010年
  - 日本IBM Innovate 2010 講演 (アジャイル・トラックの企画支援)
  - ソフトウェア品質管理シンポジウム 2010 ワークショップ提供
  - 日経BP X-over Development Conference 2010 講演
  - Agile フォーラム in 岐阜 2010 講演
- 2011年
  - Agile Japan 2011 講演
  - Agile Conference Tokyo 講演
  - ソフトウェア品質管理シンポジウム 2011 講演とワークショップ提供
- 2012年
  - Agile Japan 2012 講演
  - 電子情報産業協会 JEITA ソフトウェアエンジニアリング技術ワークショップ2012 講演
- 2013年
  - SODEC 専門セミナー 講演
  - IBM Innovate2013 The Technical Summit (Orlando, FL) 講演
  - Agile Conference Tokyo 講演

# 弊社アジャイル開発の足跡

- ~2007年3月:  
岐阜県産業振興・再開発プロジェクトに参画
- 2007年3月~:  
岐阜県下のIT企業様と生産技術力向上を目指したNPO団体(BCA)の設立(発起人)に関わり、同会員企業となる。  
現在;弊社社長戸田孝一郎が理事長
- 2007年4月~2008年3月  
同NPOの初のアジャイル開発プロジェクトをPM管理者(スクラムマスター)として実施
- 2007年6月~2008年3月  
同NPOのアジャイル研修プログラムを開発
- 2008年4月~  
Scrum&XPでのアジャイル開発に関するプロジェクト管理の指導育成  
アジャイル・チーム育成コーチング と アジャイルトレーニング の提供
- 2010年8月~  
Scrumのプロセスにユーザーストーリー(マイク・コーン提唱)を組み合わせる
- 2011年10月~  
UX(ユーザーエクスペリエンス)とアーキテクチャ設計(ACDM)を組み合わせる
- 2012年1月~  
データベース・リファクタリングを検証
- 2012年8月~  
トヨタのTMS思想を全面的に採用し、Scrumプロセスの運用に組み込む
- 2013年1月~  
豊田マネジメント研究所(TMSの専門指導)との協業でITマネジメント塾(カイゼン塾)を指導  
すりあわせ開発を提唱

# アジャイル開発の指導・支援実績

- 会計パッケージ開発
  - アジャイルによるパッケージ機能拡張
  - 発注会社:受注会社間の遠隔プロジェクト遂行
- 生産管理システム
  - ハード&ソフトの並列同時開発
- マッシュアップ&アジャイルによる地域ICTの実現
  - 買い物弱者支援システム
    - 総務省の標準モデルに採択(2011年7月)
- 保険金融系情報企業のアジャイル化支援
  - 複数のチームを並行支援指導
  - Scrum Master育成支援
- 携帯向けコンテンツ企業のアジャイル化支援
  - スマートフォンへの戦略的移行についてもアジャイル化支援
  - 複数事業部:複数チームの平行支援指導
  - Scrum Master育成支援
- アジャイル手法を応用した開発系以外での支援指導
  - 通信会社営業企画部門
  - ITIL適用部門での現場ボトムアップ(プロセス改善・ムダとり)
- すりあわせ開発の実証
  - 中堅企業の基幹系システム再構築プロジェクトに応札し、アジャイル開発で提案、お客様満足最大化を実証 (スクラムマスター支援)
- TMSでのIT職場活性化指導
  - 保険金融系企業、コンテンツ・プロバイダー企業、大手IT企業での指導