オブジェクト倶楽部
クリスマスイベント 2006
PFトラック1

プロジェクトメンバーを笑顔で見送るための心構え

～一期一会のメンバーと共有すべきこと～

2006.12.20

(株) 永和システムマネジメント
プログラマー
伊藤 浩一 (Koichi ITO)

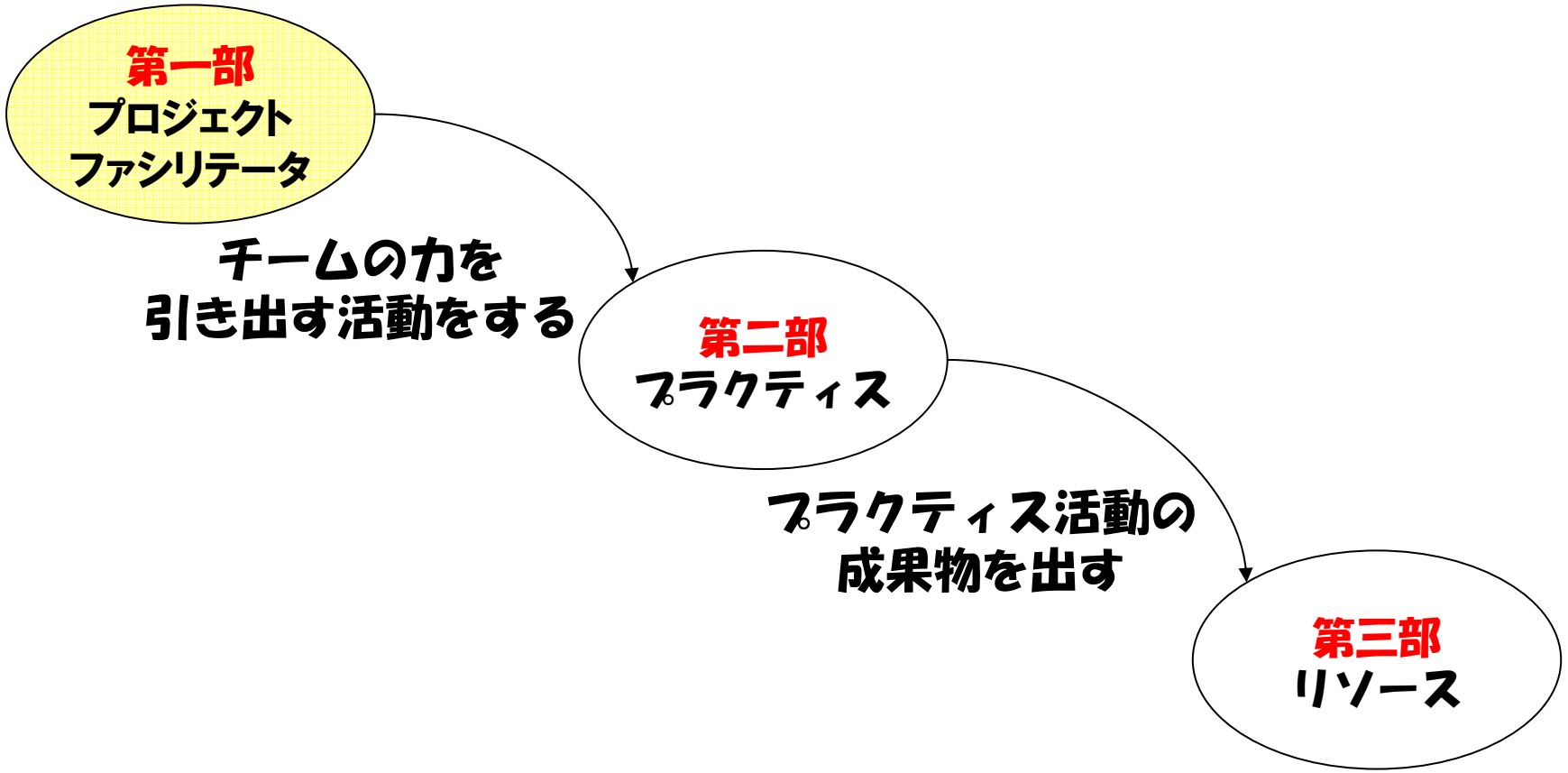
自己紹介

- 名前：伊藤 浩一
- 所属：株式会社 永和システムマネジメント
- 主な仕事：
  - 昼：Javaによるワークフローエンジンの開発
  - 夜：RubyやRuby on Railsに関する原稿書き
- 好きなプラクティス：
  - デベロッパーテスティング
  - 計画ゲーム

目次

- 第一部 プロジェクトファシリテーション再考
- 第二部 プロジェクトとプラクティス
- 第三部 プロジェクトに残り続けるもの

ロードマップ


第一部
プロジェクト
ファシリテータ
チームの力を
引き出す活動をする
第二部
プラクティス
プラクティス活動の
成果物を出す
第三部
リソース

第一部

プロジェクト
ファシリテータ
再考

プロジェクトファシリテータの目的

QoEL向上の実現

プロジェクト ファシリテータの 役割

(1)プロジェクトの場作り

(2) メンバーの持っている力を引き出す

プロジェクトファシリテータは人？

プロジェクト ファシリテータは ロール

ロールを
担えるもの

プロジェクトの全参加者

人と
コンピュータ

人とコンピュータの特徴

- 人
  - 創造的な作業ができる (本来頭脳を使うべきこと)
    - 人は楽しいことが好き
  - 同じ仕事を繰り返し正確に行うことが苦手
    - 単調、複雑な繰り返しは得てしてストレスに感じる
- コンピュータ
  - 創造的な作業が苦手
  - 同じことを繰り返し正確に行うことが得意

得意な領域が異なる

誰が楽しく
あるべきか？

もちろん人

(Not Computer!)

プロジェクトに
コンピュータを積極的に
参加（活用）する

コンピュータを活用すべき所

- 注意を払う必要のある作業
  - プロジェクトでの落とし穴
- 心の折れる作業
  - 単調な繰り返し作業
  - 複雑な手順を踏む作業

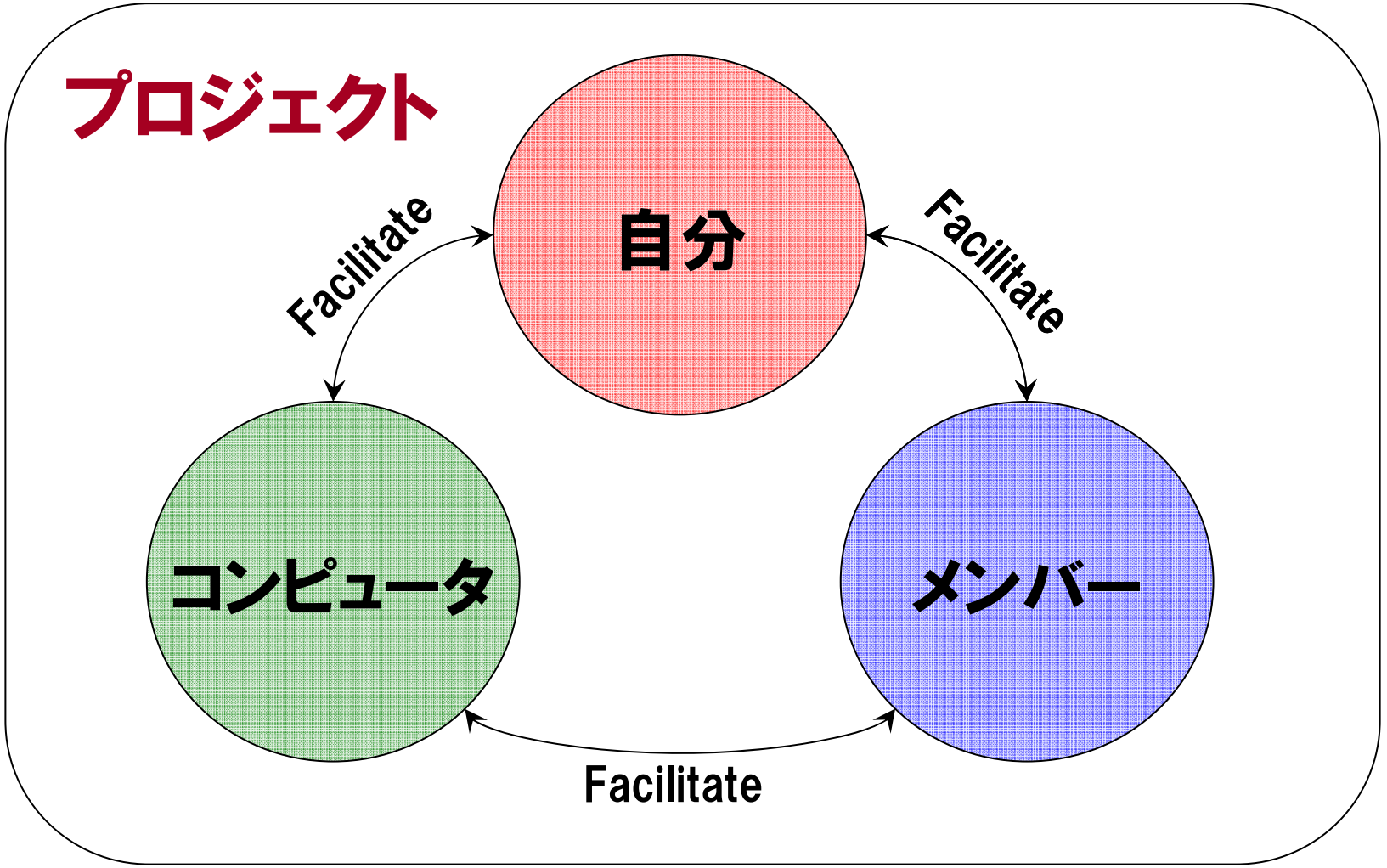
人がストレスを感じる作業は
なるべくコンピュータに割り当てる

第一部
まとめ

すべてのメンバーが
プロジェクトファシリテータ
であるべき

マルチプロジェクト
ファシリテータ

ファシリテートの内回り・外回り



ロードマップ

第一部
プロジェクト
ファシリテータ

チームの力を
引き出す活動をする

第二部
プラクティス

プラクティス活動の
成果物を出す

第三部
リソース

第二部

プロジェクト
と
プラクティス

プロジェクトに
プラクティスを
適応させること

環境の変化

- プロジェクトメンバーの変化
  - システム専門知識の低下
  - プロジェクトの文化への慣れ
- 開発状況の変化
  - 新規開発期
  - メンテナンス期

メンバーは環境の変化に適応する必要がある

変化の難易度

- 難易度 高
  - 人（他人）
- 難易度 中
  - 人（自分）
- 難易度 低
  - プラクティス（仕事術）
  - コンピュータ（ハードウェア、ソフトウェア）

今回お話しする対象

どのように変化していったか？

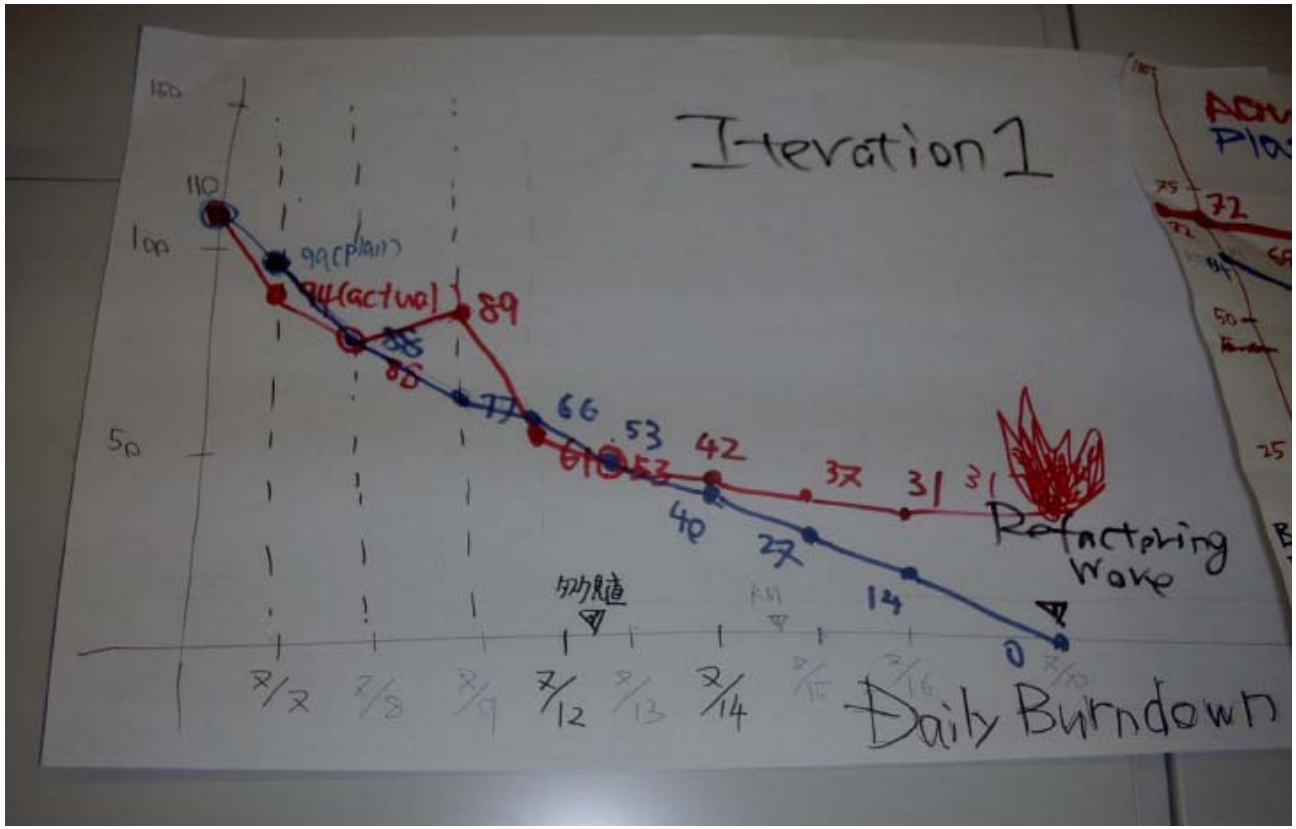
プラクティスを改善する

- プロジェクトは有名プラクティスを実践すれば良くなるものではない
  - プラクティス実践のコスト
  - メンバーのモチベーション
- 実践されなくなったプラクティス
  - バーンダウンチャート
  - にこにこカレンダー
  - ふりかえり

今回取り上げる変化例

バーンダウンチャートの変化

バーンダウンチャート


Iteration 1
110
100
50
99(plan)
89
66
53
42
37
31
40
27
14
0
Refactoring Week
勉強
7/2
7/8
7/12
7/13
7/14
Daily Burndown

バーンダウンチャートの概要

- アジャイル界での有名プラクティスのひとつ
  - 本邦にはかくたに氏が輸入（XP祭り 2004で発表）
  - 縦軸にタスク量、横軸に時間を表し、タイムボックスの中の残タスク量を『見える化』するプラクティス

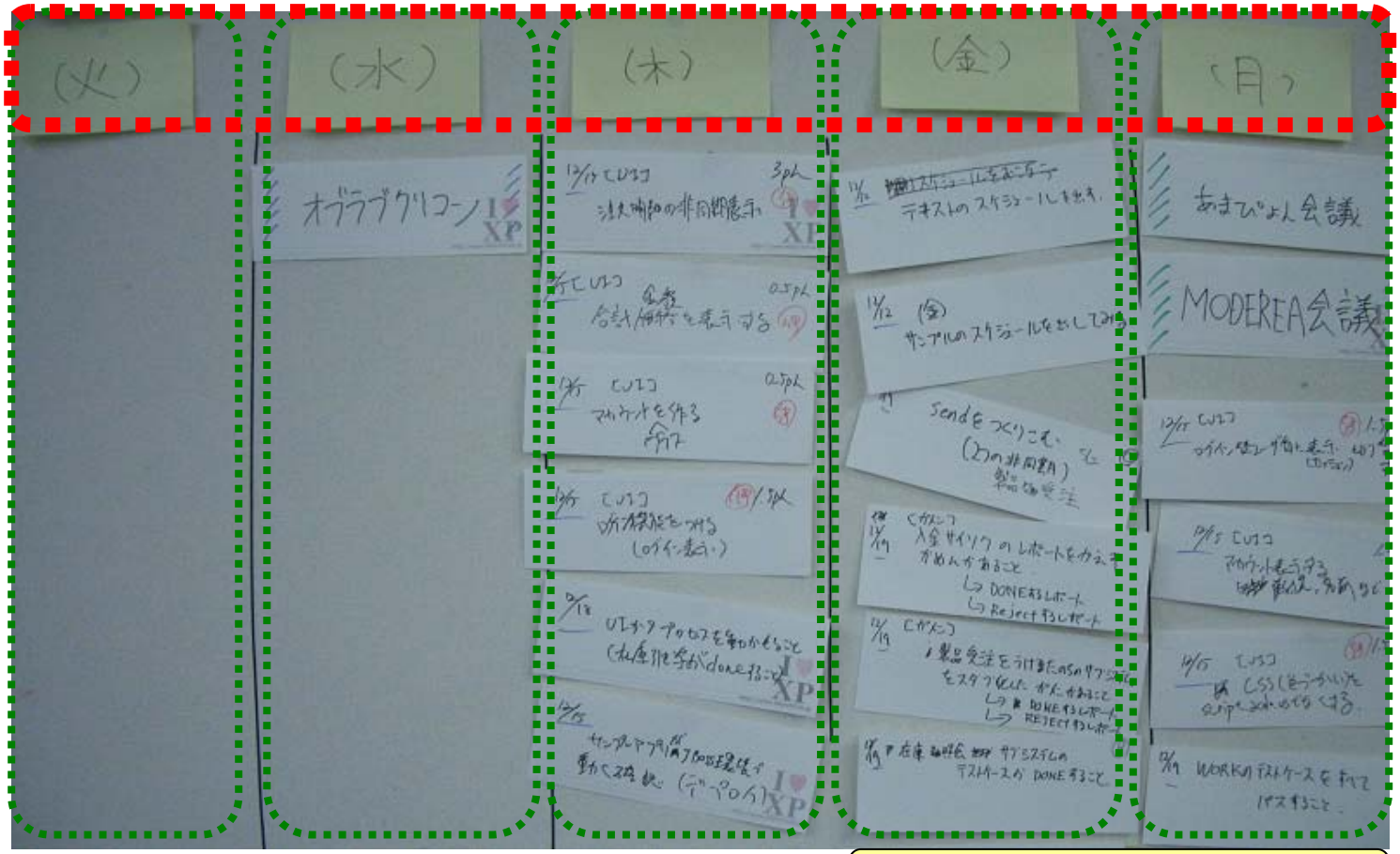
バーンダウンチャートをやめた理由

- メンテナンス期になり更新されなくなった
  - 一週間のタイムボックスの中でも、稼動しているシステムからの要求でタスク内容が激変する
  - 旧来どおりの表現では一週間のタスク管理がしづらくなった

バーンダウンチャートの変化

献立（仮名）

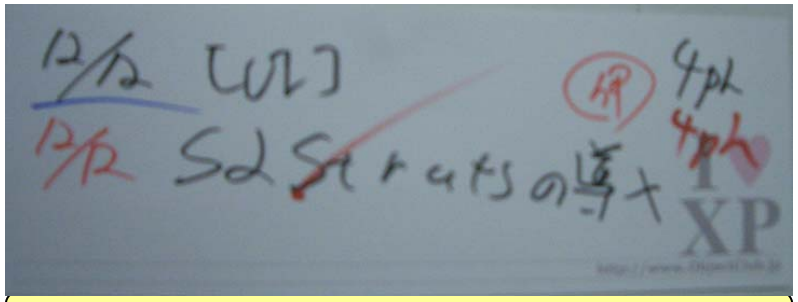
献立（仮名）



2006年12月19日撮影

曜日　曜日単位のタスク

(補足) タスクカード



タスクカード

- 材料
  - 大きすぎない紙
    - 13cm×18cm (の半紙)
- レシピ (タスク完了まで)

1. 材料の紙を半分に裁断
2. 計画を立てた日、タスク内容、見積り時間を書く
   - タスクのサイズはカードに収まる粒度にする
3. サインアップする
4. 完了後実績日時を書く

献立 (仮名) について

- 連絡ひとつで作業が切り替わる状況に対応
  - タスクかんばんにバーンダウンチャートを足したようなもの
  - 一週間のタイムボックスの中での残タスクの見える化
    - 一週間の中で行うタスクを曜日単位に貼り付ける
    - 時間単位だと計画の粒度が細かかった
  - タスクカードを容易に貼り替え可能
    - 作業内容の変化に応じて常に壁面を更新する
- タスクかんばんの変化発展版

ペアプロについて

2006年11月28日付の中の人の日記

伊藤さんの方もプロジェクトの経験をフィードバックするお話。こちらはXPバリバリのプロジェクト(あ、でもペアプロは見たことないな)から得た知見をお話いただけるのではないかと思います。

ペアプロと
ソロプロ

ペアプログラミング

- メリット
  - 知識共有ができる
  - 常時レビューがされる（間違いにすぐ気づいて直せる）
  - etc…
- デメリット
  - 作業の瞬間最大速度の低下

一度身に付けると
身体が覚えているプラクティス

ソロプログラミング

- メリット
  - 作業の瞬間最大速度の向上
- デメリット
  - 知識共有の場を別途設ける必要がある
  - 常時レビューされない（間違いにすぐ気づかない）

不安なときは常に
レビューを依頼する態度を持つこと

現在の姿はペアワーク

- プログラミングに特化しない
  - 計画ゲーム、仕様検討、設計議論など
    - 人と会話することにより頭にあることを吐き出してまとめられたり、刺激を受けて新たな気づきを得ることができる
- プログラミングをペア行うとき
  - 知識の共有やスキル転移をしたいとき
  - 一人だと心が折れそうなとき

状況に応じてペアで行う作業と
ソロで行う作業を見極めるのが大事

変わらず価値を提供する実践

常に有効なコアプラクティス

- テスト駆動開発
  - 開発の基本スタイル
  - デベロッパーテスティングは開発の足場
- 朝会
  - 定期的なプロジェクトやメンバーの状態を情報共有する場
- プロジェクト自働化
  - テスト駆動開発と並び開発の足場となる
  - 後々まで人が楽をするため
    - 自動化のコストはそれなりにかかる
  - コンピュータと作業を住み分けるため

第二部
まとめ

プロジェクトに
プラクティスを
適応させること

ロードマップ

第一部
プロジェクト
ファシリテータ

チームの力を
引き出す活動をする

第二部
プラクティス

プラクティス活動の
成果物を出す

第三部
リソース

第三部

プロジェクトに残り続けるもの

プロジェクトに残るものは
動くコードと
メンテされたドキュメント

動くコードとは?

- プロダクトコード
- テストコード
- 自働化スクリプト

ものには、すべて名前がある

「静的解析とはつまりソースコードの解析だ。そしてソースコードの解析とは名前の調査である。ファイル名・関数名・変数名・型名・メンバ名など、プログラムは名前のかたまりだ。」

-Rubyソースコード完全解説（青木 峰郎）

名前重要

- 動くコードの中で最も重要な要素のひとつ
- プログラミングとは名前を付ける作業
- 既存システムを解析中、最も遭遇する有力な手掛かり
  - 名前付けはペアプログラミング慣れしていないメンバーも参加しやすい

テストについて

Code without tests is bad code. It doesn't matter how well written it is; it doesn't matter how pretty or object-oriented or well-encapsulated it is. With tests, we can change the behavior of our code quickly and verifiably. Without them, we really don't know if our code is getting better or worse.

-WORKING EFFECTIVELY WITH REGACY CODE
(Michael C. Feathers)

重要なのはテストと共にあること

動くソフトウェアを残す習慣

- 常に動くコードの共有を意識付ける
- バージョン管理ツールへのコミットサイクル
  - テストコード（赤）をパスするプロダクトコード（緑）を書いたらコミット（Red → Green）
  - テストコード（緑）をパスするプロダクトコードをリファクタリングしたらコミット（Green → Green）

動くソフトウェアを
短い周期で管理、共有する

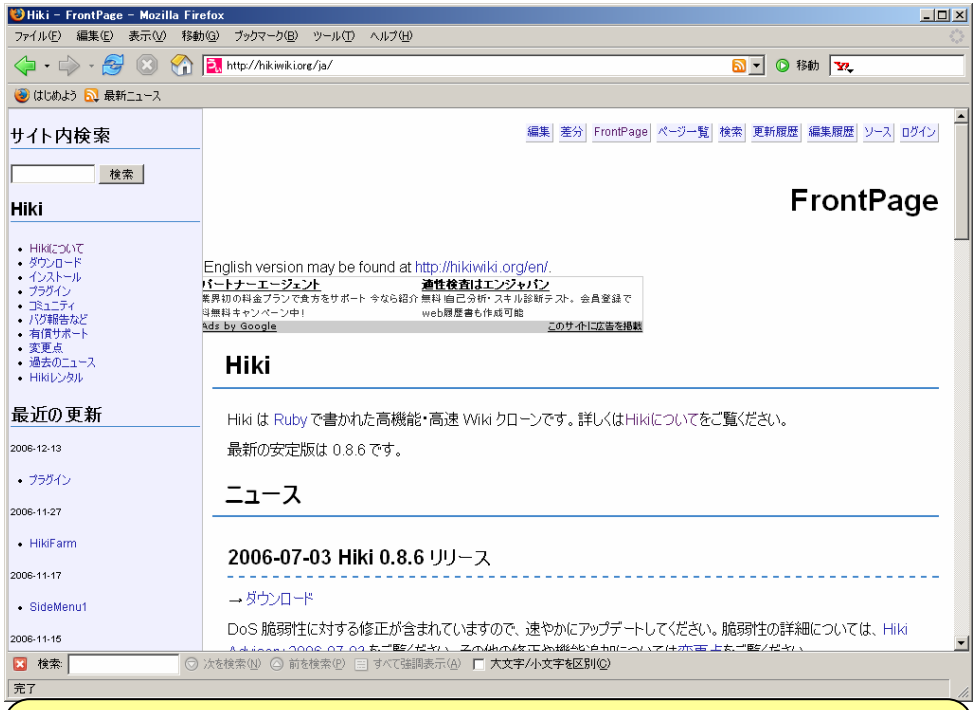
自働化スクリプトについて

- **誰でもリリース可能**にするため、リリースまでの作業をできるだけ自働化することを心掛ける
- 自働化しやすいこと
  - ビルド、テスト、アセンブリ、デプロイなどの定型作業
    - これらはMaven、Ravenなどの既存ツールを利用できる
- 一気呵成の自働化は難しい
  - 気づいた人が日々コツコツと行う習慣が大事
  - すぐ自働化できないことはドキュメント化やToDoタスク化する

ドキュメントについて

- ドキュメント化する物事
  - 自動化されていないビルドプロセスの手順
  - プロジェクトでのハマリ所、落とし穴
  - 設計意図を伝えるもの（e.g. 議事録、開発ノート…）
- 注意点
  - システムとの同期が必要なドキュメント（e.g. 操作マニュアル…）は考古学資料を作らない心掛けをする
    - メンテナンスされていないドキュメントはただの考古学資料
  - メンテ可能なドキュメント量

日常的な情報共有整理



Hiki…http://hikiwiki.org/
Rubyで書かれたWikiエンジンクローンの一つ

- Wiki
  - 困ったときの検索機能
  - ドキュメント共同所有
  - だれでも、いつでも気軽に手軽に編集できる

鮮度の落ちやすい資産

- テストコード
- 自働化スクリプト
- ドキュメント

メンテを習慣付ける
躾が大事

第三部
まとめ

プロジェクトに残るものは
動くコードと
メンテされたドキュメント

まとめ

本日伝えたかったこと

- QoEL向上のため自働化を心がける
- プロジェクトの性質や状況に合わせたプラクティスを心がける
  - テンプレート（プロセス）にプロジェクトを合わせるのではない
- 動くコードとメンテしたドキュメントを残すことを心がける

謝辞

- 参加者の皆様
- オブLOVEの中の人
- 家永さん
- かくたにさん
- t-wadaさん
- 猪狩さん
- 右近さん
- KKDさん
- 須藤 功平さん
- チームかくたにの卒業生のみなさん

Enjoy Engineering Life

ご清聴
ありがとうございました