Development Baseline

Archway Inc.

Consulting Service

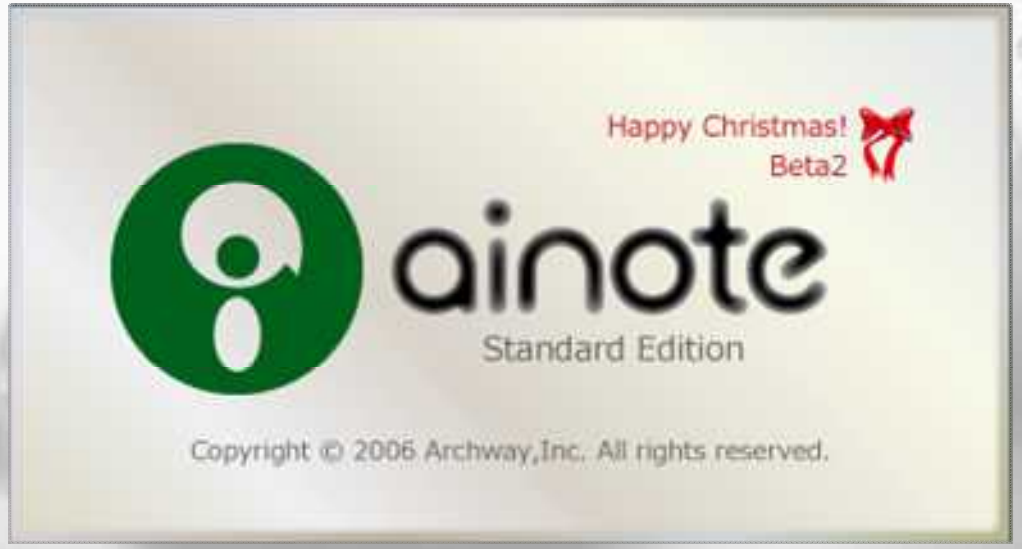
http://www.archway.co.jp/

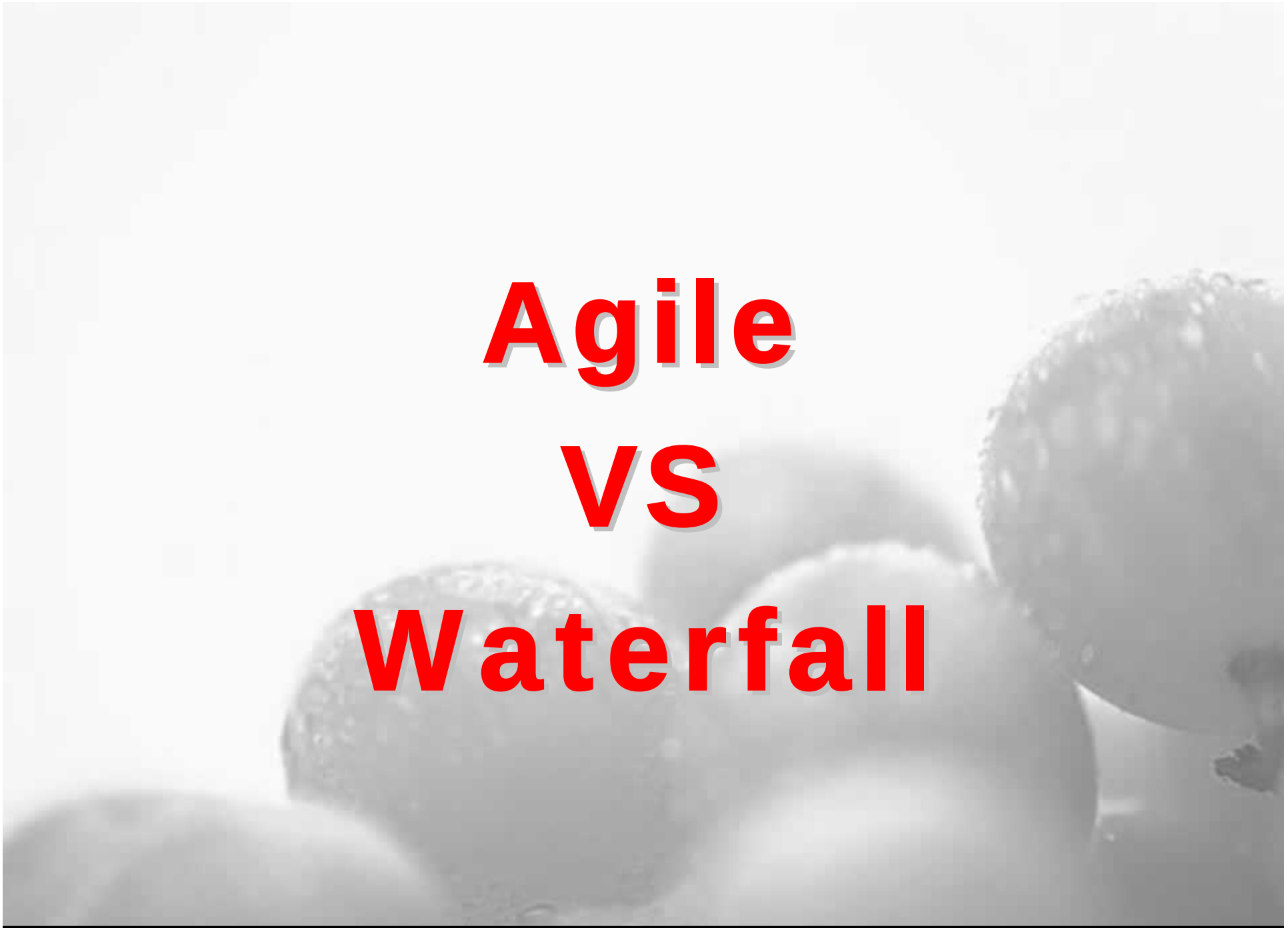
自己紹介

- 中西 庸文(**Tsune**fumi Nakanishi)
  - 株式会社アークウェイ大阪事業所 所長
  - コンサルではなくて**メンター**を目指している。
    - 一歩前から業界を引っ張っていくのは優秀な方々におまかせして、僕は一歩後から業界を**底上げ**していきたい。
- E2P
  - **E**ros **E**ros **P**rince
    - なんだか知らないうちに**a-fukui**さんに名前付けされた。
    - 実はそんなにエロくないのに。
    - 特技は**カンチョー**
    - カンチョーは**日本の文化**です。

aiNote

- フィーチャ モデリングツール
  - クリスマスにBeta2リリース
  - http://www.archway.co.jp/





What The Fuck Is That?

- AgileはWaterfallを敵にしないと正当性を証明できないほど**無力**なのだろうか？
- Waterfallの**歴史**から学ぶべきところはないのだろうか？
- プロジェクトがうまく回らないことに対する**言い訳**に開発方法論を持ち出していないだろうか？
- とは言うものの、そもそも開発方法論の**適用**がそんなに大事なことなのだろうか？

ドメイン特化方法論

- 本当に大事なのは、**私たちの開発チーム**というドメインに特化した方法論。
- ベースの方法論はAgileになるかもしれないし、そうじゃないかもしれない。
- 重要なのはベースの方法論ではなく、自分たちで**ふりかえり**を行い、**KAIZEN**した結果として最適化された**持続可能なプラクティス**の集合があるかどうか。
- 自分たちの**文化**に適合させなければ続かない。

もっと自信を

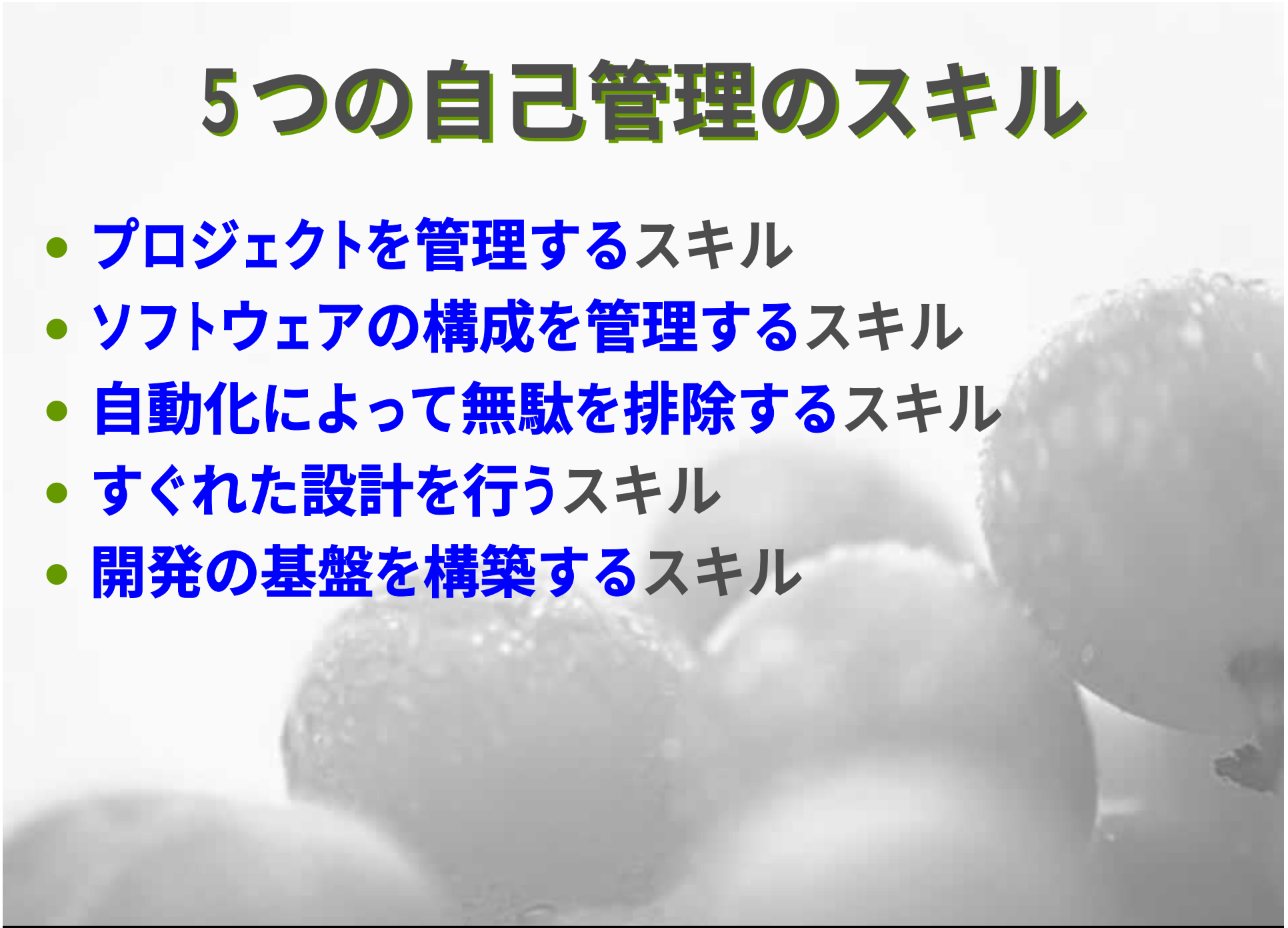
- 自分たちのチームにとってベストプラクティスならそれは**正解**。
- **やるべきこと**をきっちりこなしていれば、自信は後からついてくる。
- 自信があれば、他者を**非難すべき理由**などあるはずもない。
- ただし、自分をいつもしっかりと**コントロール**していなければ自信は逃げていってしまう。

自分たちのために管理する

- 実は他人に管理**されている**方が楽。
- でも本当に管理すべきなのは**自分自身**。
  - 自分自身の**スキル**を管理する。
  - 自分自身の**コミュニケーション能力**を管理する。
  - 自分自身の**姿勢**を管理する。
  - 自分自身の**無駄**を管理する。
- 自分たちの管理もできていないのに**ソフトウェア開発**を管理できますか？

5つの自己管理のスキル

- プロジェクトを管理するスキル
- ソフトウェアの構成を管理するスキル
- 自動化によって無駄を排除するスキル
- すぐれた設計を行うスキル
- 開発の基盤を構築するスキル



目新しいことなど何もない

- ソフトウェア開発において、やらなければいけないあたりまえのこと。
- あたりまえのことがきっちりできていますか？
- 本当の気づきとは既知であると思っていた基本を見つめなおす作業の過程で得られるもの。
- うまくいかない理由に対する言い訳を探すよりもまず地に足のついた開発が行えているかどうかをふりかえるべき。

変化するためにまず決める

- 何かを**決定**しないと、変化することはできない。
- 決めて実行すれば、変えなければいけない**本質**もおのずと見えてくる。
- スタートからゴールまでにやるべきこと(**目的**)を明確にしてから、その実現方法(**手段**)を導き出す。

Development Baselineとは

- Development Baselineとは、ソフトウェア開発のライフサイクルにおいて**真にやるべきこと**から**最適な実現方法**を決定すること。
- ただし、ツールに縛られてはいけない。**目的に縛られるべき**である。
- 時として、目的に合わなくなったお気に入りのツールを**捨てる勇気**も必要。

Development Baselineでやるべきこと

- プロジェクト管理
  - (**P**roject **M**anagement)
- ソフトウェア構成管理
  - (**S**oftware **C**onfiguration **M**anagement)
- 自動化
  - (**P**roject **A**utomation)
- テスト駆動開発
  - (**T**est **D**riven **D**evelopment)
- ゼロ機能リリース
  - (**Z**ero **F**eature **R**elease)

プロジェクト管理(PM)

- **タイムボックス管理**
  - フィーチャ プランニング
  - リリース プランニング
  - イテレーション プランニング
- **進捗のトラッキング**
  - 要望
  - タスク
  - バグ
- **情報の共有**
  - プロジェクトに関する情報を一個所に

ソフトウェア構成管理(SCM)

- リポジトリ
  - メインライン
  - リリースライン
  - 実験用ワークスペース
- プライベート ワークスペース
  - 自分の作業場所を分離して変更を管理
  - マージを受け入れる勇気
- タスクレベルのコミット
  - 粒度の細かいタスクが完了するたびに、こまめにコミットする癖をつける

自動化(PA)-自動ビルド

- **1ステップビルド**
  - 毎回1ステップでソフトウェアをゼロからビルド
- **移植可能なビルド**
  - どのワークステーションにおいても同じ結果となるビルドの生成
- **結合に対する自信**
  - 成功するビルドを簡単に生成できることで、結合時の不安を解消
- **ビルド手順の明確化**
  - 手作業のビルドで発生しがちな単純なミスを解消

自動化(PA)-継続的インテグレーション

- **定期ビルド**
  - リポジトリの変更を検出して自動ビルド
  - スケジューリングによるビルド
- **詳細なレポート**
  - ビルド結果
  - テスト結果
  - コードカバレッジ
  - コーディング規約
  - コードの重複
  - etc..
- **ビルド結果通知**
  - Emailによる通知
  - Extreme Feedback Devicesによる通知

テスト駆動開発(TDD)-受け入れテスト駆動

- **ソフトウェアの内部構造と分離されたテスト**
  - 顧客の要求の実現とソフトウェアの内部構造は分離すべき
- **要求ドリブンによる進捗管理**
  - 失敗する受け入れテストを先に作成し、成功した受け入れテストの数がそのまま進捗に。
  - ビジネス価値に注目した進捗管理

テスト駆動開発(TDD)-単体テスト駆動

- 開発のハートビート
  - レッド
    - 新しい機能に対してテストを書く
  - グリーン
    - テストが成功するような最低限の実装を行う
  - リファクタリング
    - テストが通る状態のまま、内部構造の改善を行う
- 設計手法の確立
  - テスト駆動開発はテスト技法ではなく設計手法
  - 目標は、動作するきれいなコード
- 自己テストコードとしての再利用
  - テスト駆動開発において作成された単体テスト群は自動ビルド時に優れた自己テストコードとなる

ゼロ機能リリース(ZFR)

- **基盤の構築**
  - テスト基盤
  - 自動化基盤
- **フィーチャの抽出**
  - メイン フィーチャ
  - ハイリスク フィーチャ
- **アーキテクチャ プロトタイプの作成**
  - メイン フィーチャからアーキテクチャのプロトタイプを作成
  - ハイリスク フィーチャを検証
- **開発者トレーニング**
  - 躾(自分たちのことは自分たちで)

- **これらの作業に専念し、機能を何も実装しない最初のイテレーションがプロジェクトの成功を大きく左右する。**

Development Baseline実装例