企業システムにアジャイルは必要か

"Agile" is really need to enterprise system?

2015/6/22

株式会社ゼンアーキテクツ

岡　大勝

ZENARCHITECTS

自己紹介

岡　大勝

（おか　ひろまさ）

株式会社ゼンアーキテクツ

CEO/チーフアーキテクト

プロフィール

- 第一勧銀情報システム（現：みずほ情報総研）
  - VOS3 COBOL＆MS-Cプログラマ
- 日本ディジタルイクイップメント（日本DEC）
  - 主に生保・損保・銀行向けの資産運用システムに携わる。
  - Alpha NT, SQL Server, DECnet
- 日本ヒューレット・パッカード C&I-Financial
  - 金融機関向けのシステムアーキテクチャ設計、開発プロセス設計、運用プロセス設計を行う。
  - HP-UX, J2EE, WebLogic, Oracle Database, OO
- 日本ラショナルソフトウェア
  - 開発プロセス（RUP）およびオブジェクト指向分析設計手法の導入コンサルティング。
  - RUP, Rose, ClearCase, Functional Tester
- ゼンアーキテクツ
  - 2003年よりIT設計事務所として活動。お客さまのIT投資の最適化を目指し、アーキテクチャ中心のプロセス改善を推進。

著作/翻訳

- 「要求」の基本原則（技術評論社）2009
- 本当に使える開発プロセス（日経BP社）2012
- ディシプリンド・アジャイル・デリバリー（翔泳社）2013

そもそも「アジャイル」は必要なのか？

- ウォーターフォールで何が悪い？
- いや、悪くない
  - QCDを最初に厳密に規定し計画するのがウォーターフォール。
  - 計画通りにプロジェクトが完了するのであれば、むしろ望ましい
- なぜ、別の手法が必要なのか
  - プロジェクトが「最初に規定し計画した通り」に進まない。
- なぜ、計画通りに進まないのか
  - システム開発には、不確実性の高い要素が多い＝“リスク”
  - 「要求リスク」「スキルリスク」「技術リスク」「政治リスク」

できるだけ早い時点でリスクをキャッチアップしながら
プロジェクトを進める手法が必要

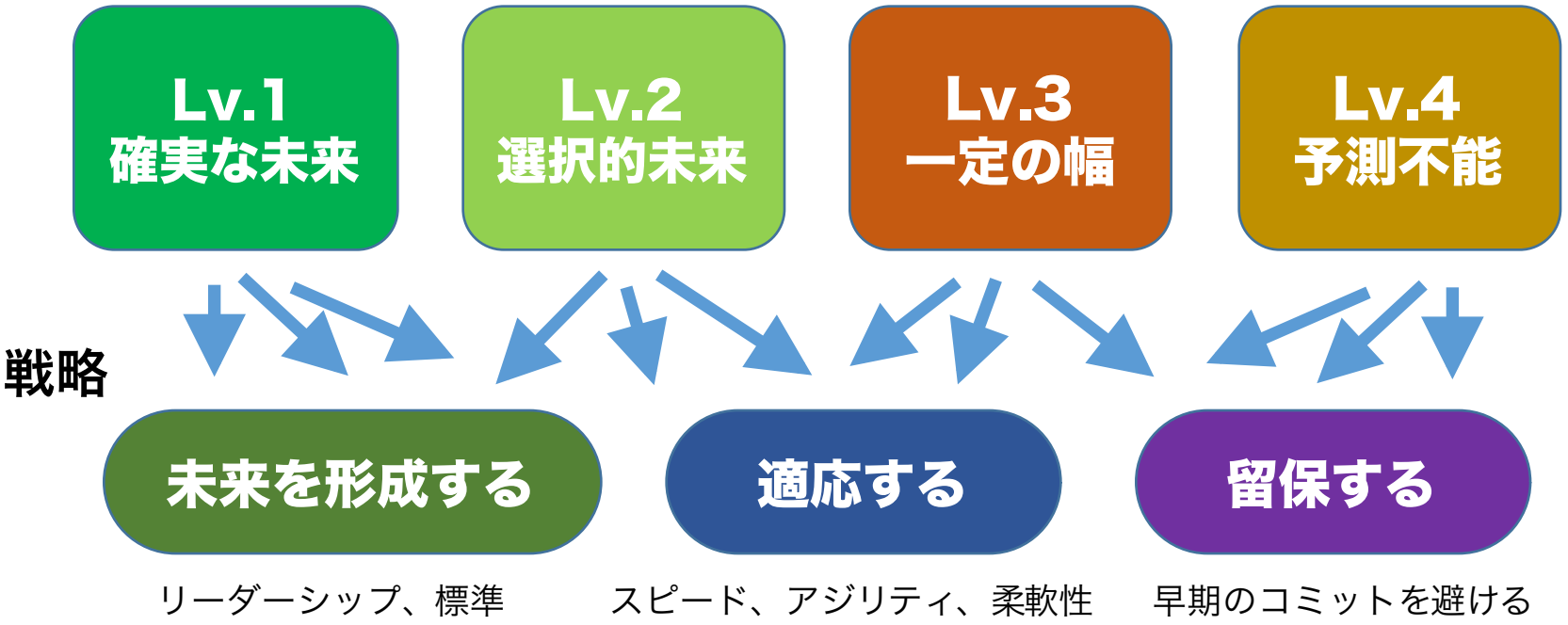
© 2015 ZEN ARCHITECTS Co.,Ltd.

アジャイルの期待効果１
«不確実性への取り組み»

© 2015 ZEN ARCHITECTS Co.,Ltd.

”不確実性”のレベル

- ほとんどの事象は想定可能な一定の幅に収まる
  - ビジネスでは、Lv.1=50%、Lv.2+Lv.3=50%、Lv.4=まれに出現



「不確実時代の行動と戦略」ヒュー・コートニー、他
ハーバードビジネスレビューブックス：不確実性の経営戦略（ダイヤモンド社）より

ZENARCHITECTS

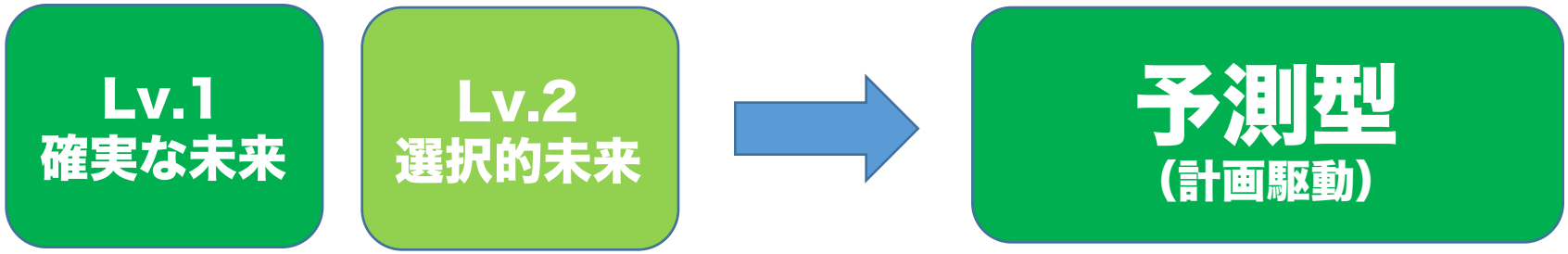
© 2015 ZEN ARCHITECTS Co.,Ltd.

計画できることは、計画して実施する

- 「正しい」ことが分かっている
  - よほど大きな問題が発生しない限り“うまくいく”
  - 事業者は「必要な時期」に「必要なもの」が手に入れば良い。

- 定型反復業務
- 確立された手順

PMBOK 5th



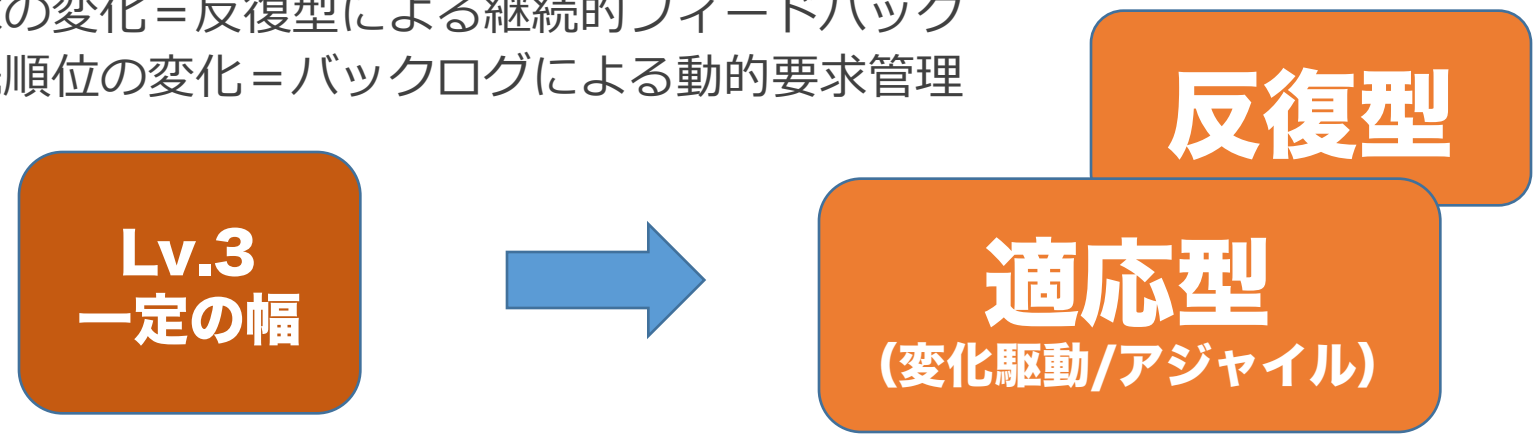
© 2015 ZEN ARCHITECTS Co.,Ltd.

計画できないことは、確認しながら進む

- 何が「正しい」のか判断できない
- 正しいものが変化する
  - 機能
  - 技術
  - ビジネス、マーケット

- 変化への継続的な対応
- 「要求の変化」と「優先順位の変化」
  - 要求の変化＝反復型による継続的フィードバック
  - 優先順位の変化＝バックログによる動的要求管理

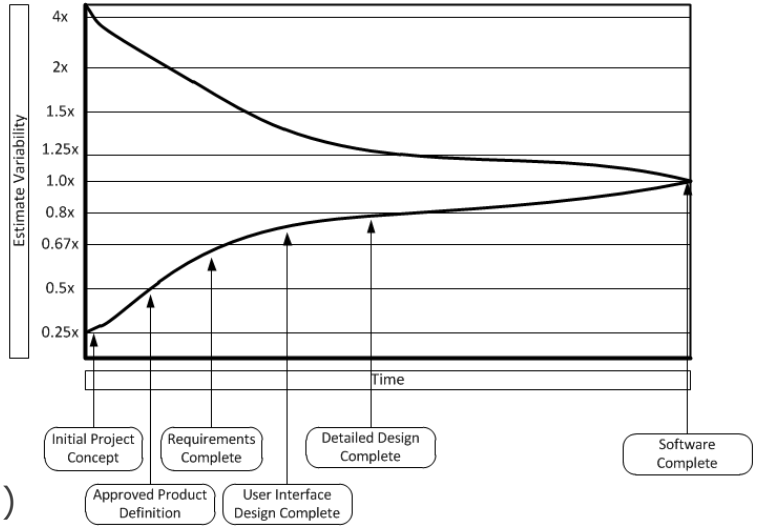
PMBOK 5th



© 2015 ZEN ARCHITECTS Co.,Ltd.

システム開発に潜む不確実性（リスク）

- 要求リスク
  - 当初想定からの機能の増加（スコープの拡大）
  - 要件の認識不足（手戻り、再作業）
  - 仕様の決定が遅れる
  - 不適切なUI
  - 顧客による継続的な変更要求
- 技術リスク
  - 想定していた性能が出ない
  - 想定される動作をしない（接続できない、想定外のエラー）
  - 目的に合致しない技術（拡張性、接続性）
- スキルリスク
  - 利用技術の経験がない
  - 今回の開発プロセスの経験がない（初めての進め方）
- 政治リスク
  - 予測不能。だが想定は自由

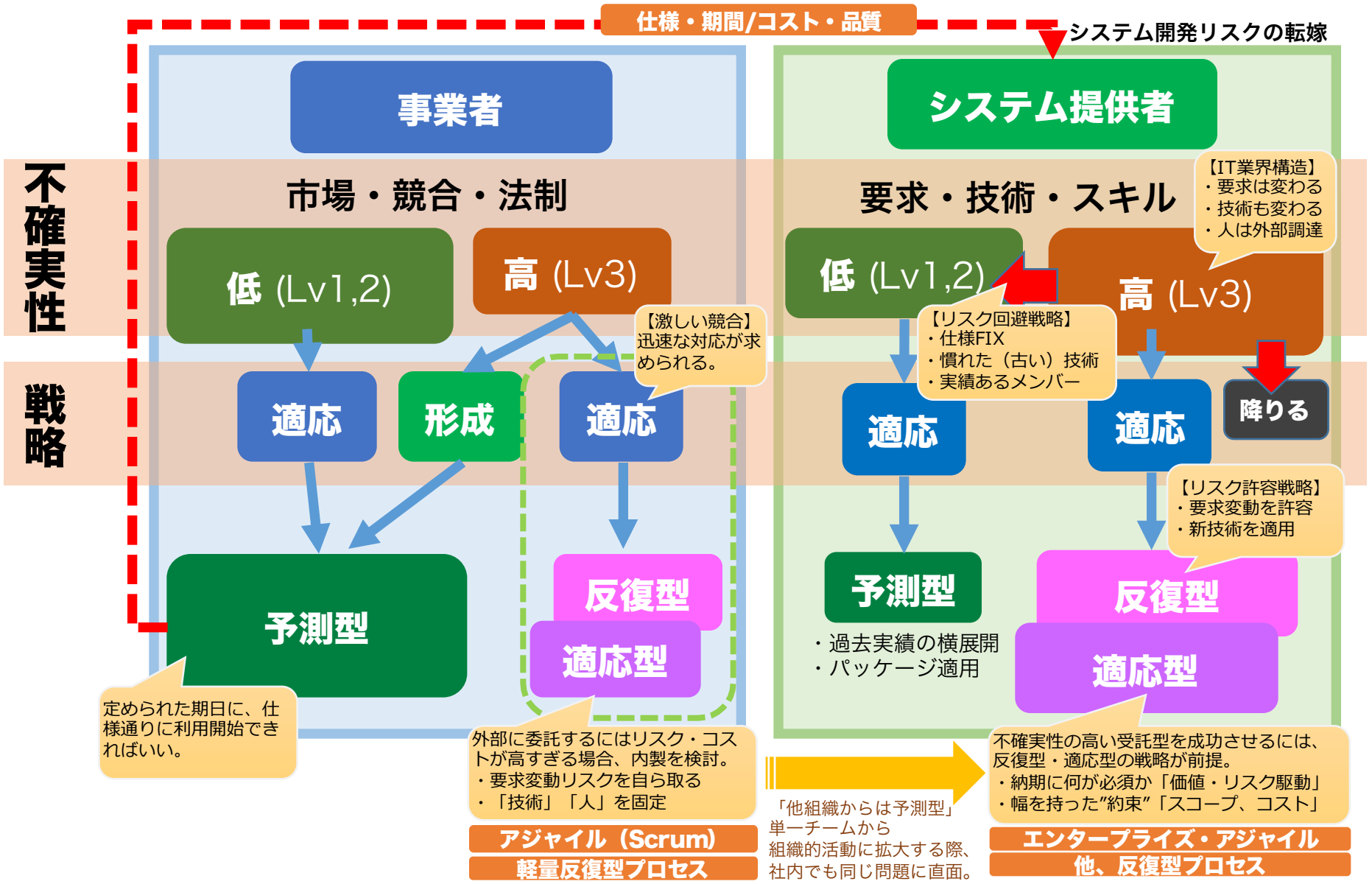


不確実性コーン(the Cone of Uncertainty)

Barry Boehm introduced to us in 1981 and Steve McConnell re-introduced in 1997 in his book *Software Project Survival Guide.*
https://msdn.microsoft.com/en-us/library/hh765979.aspx

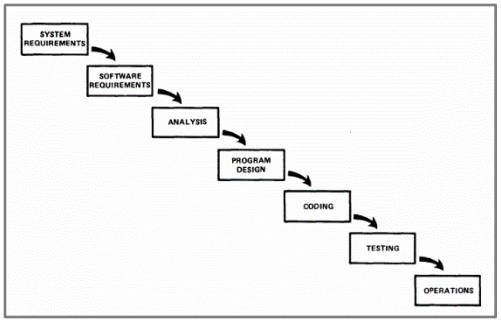
© 2015 ZEN ARCHITECTS Co.,Ltd.

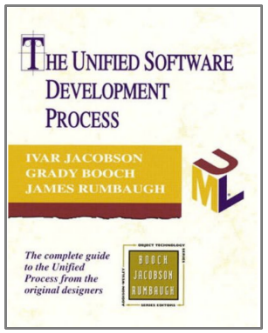
”事業者”と”システム提供者”で「不確実性」は異なる

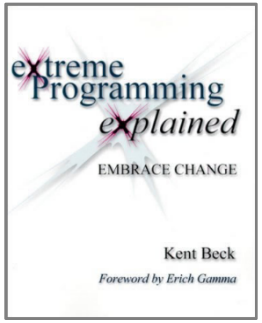


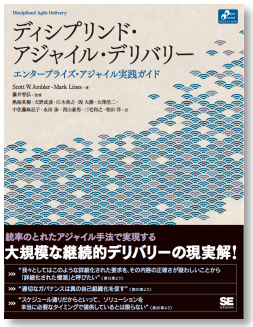
開発プロセスの歴史

- 1960年代：「Code & Go」
  - 「聞いて、書いて、動かす」
- 1970年：「ウォーターフォールモデル」Winston Royce
  - 「設計」の発明
  - 要求を正しく扱うため「分析」の導入など、その後も進化を続ける
- 1988年：「スパイラルモデル」Barry Bohem
  - 「目標」「リスク分析」「実施」「計画」のサイクルで工程が進む
- 1991年：「反復型（イテレーティブ）開発」「漸進型（インクリメンタル）開発」
  - 小さな単位で開発することで、リスクを早期に識別する。
- 1992年：「Vモデル」German Ministry of Defense
  - ウォーターフォールモデルに品質保証の観点を追加。
- 1998年：「ユニファイド・プロセス（UP）」Jacobson, Booch, Rambaugh
  - ユースケース・UMLに基づくオブジェクト指向開発プロセス。Rational社による商用版が「RUP」
- 1999年：「エクストリーム・プログラミング（XP）」Kent Beck
  - ドキュメントよりもコラボレーションに焦点を当てた軽量開発プロセス。
- 2002年：「スクラム（Scrum）」Ken Schwaber, Mike Beedle
  - 価値駆動の軽量開発プロセス。原型は1986年に竹内弘高氏、野中郁次郎氏が考案した製品開発手法。
- 2009年：「SEMAT」Jacobson他
  - これまでの作業/成果物中心のソフトウェアエンジニアリングモデルを活動/価値中心に再構築。
- 2011年：「スケールド・アジャイル・フレームワーク（SAFe）」Dean Leffingwell
  - アジャイルを単一チームから企業規模にスケールするための手法。
- 2012年：「ディシプリンド・アジャイル・デリバリー」Scott W.Ambler, Mike Lines
  - アジャイルを企業活動に適合するよう拡張。
- 2013年：「PMBOK 5th Edition」PMI
  - アジャイル（適応型ライフサイクル）がプロジェクトライフサイクル型として追加。









© 2015 ZEN ARCHITECTS Co.,Ltd.

アジャイルは何が違う？

- 進め方が違う（だけ）
  - 個々の作業は同じ
  - プロセスフレームワークの組み合わせが違う
- “計画通りには進まない“ことに対処する。
  - 動くシステムを使って（作って）みなければ、正しいかどうか分からない
  - 手直し（軌道修正）しながら進むために、作業工程を身軽にする

- 工程別フェーズ → 反復型フェーズ
- 成果物駆動 → ジャストインタイム
- 予測型（計画駆動） → 適応型（価値駆動・リスク駆動）

ウォーターフォール ”型”

アジャイル ”型”

© 2015 ZEN ARCHITECTS Co.,Ltd.

プロセスフレームワークの組み合わせ

- 開発プロセスの違いは、
  プロセスフレームワークの組み合わせの違い

  - 工程別フェーズ x 成果物駆動 x 予測型（計画駆動）
    - ウォーターフォールモデル
    - V字モデル

  - 反復フェーズ x ジャストインタイム x 適応型（価値駆動）
    - スクラム
    - XP

  - 反復フェーズ（ゴール指向） x 成果物駆動 x 適応型（リスク駆動）
    - ユニファイド・プロセス（UP）

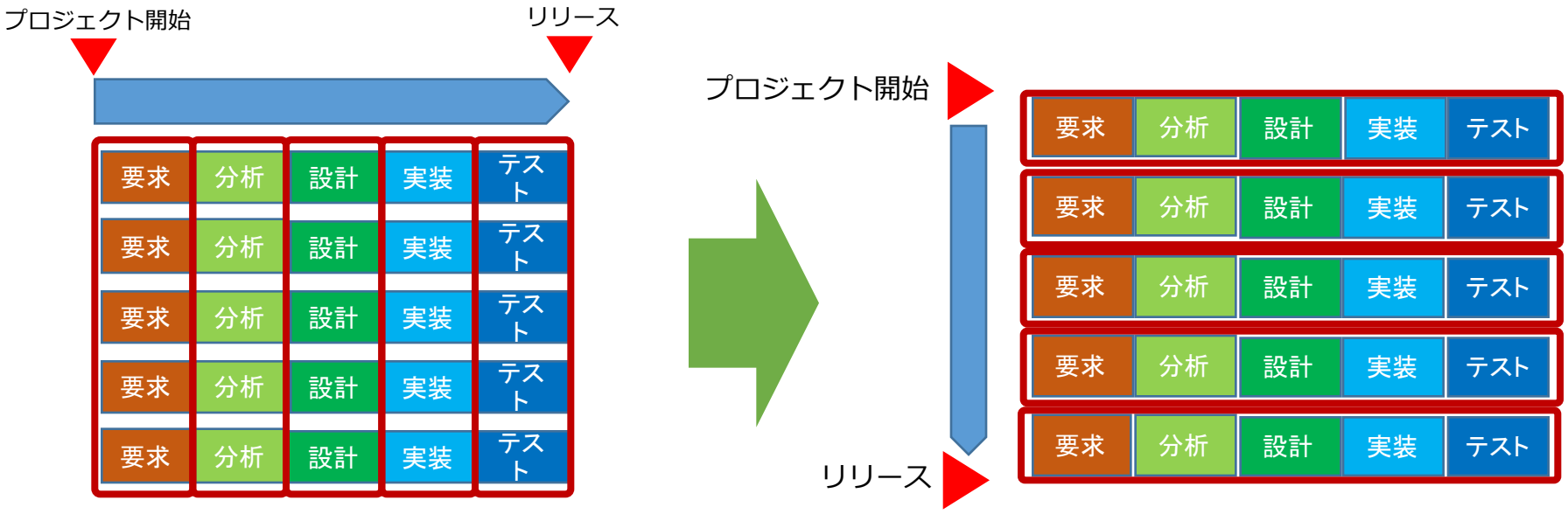
  - 反復フェーズ（ゴール指向） x ジャストインタイム x 適応型（価値・リスク駆動）
    - ディシプリンド・アジャイル・デリバリー（DAD）

© 2015 ZEN ARCHITECTS Co.,Ltd.

「反復型」フェーズ

- 要求の実装完了が、フェーズの完了基準
- 特徴
  - 動作する機能が早期に利用できる
  - 統合とテストの負担が大きい

技術リスク・要求リスクへの対処



工程別フェーズ　　反復型フェーズ

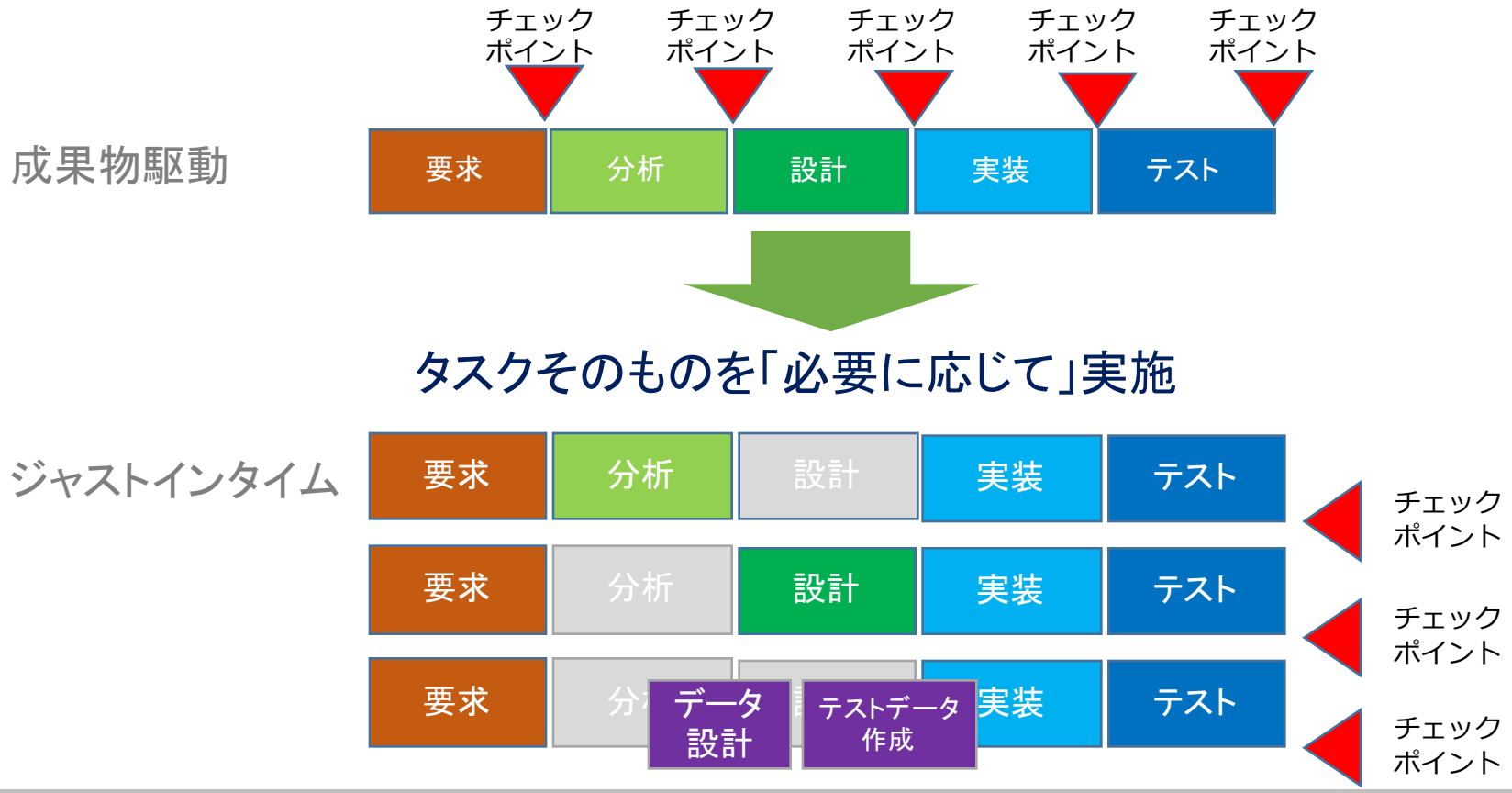
© 2015 ZEN ARCHITECTS Co.,Ltd.

アジャイルの期待効果２
«開発効率の向上»

© 2015 ZEN ARCHITECTS Co.,Ltd.

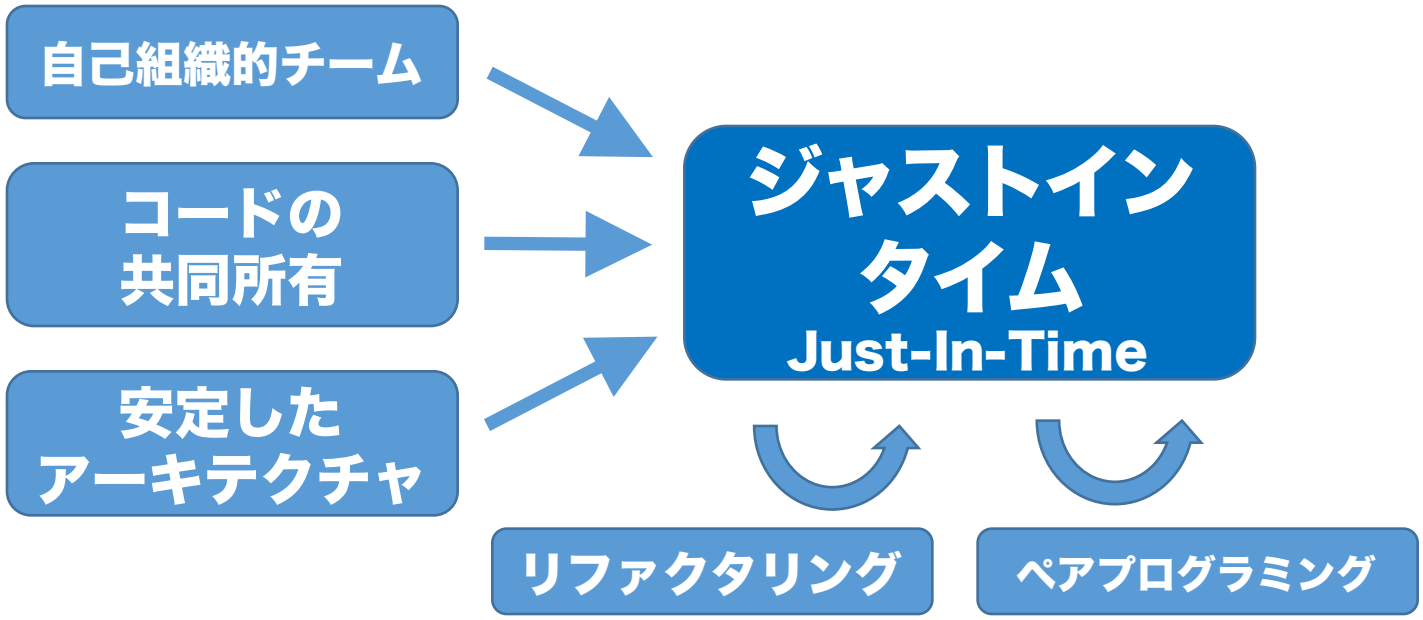
ジャストインタイム（JIT）

- 作業の管理を、テスト結果によって行う
  - 中間成果物は「それが必要な場合に」作成する



© 2015 ZEN ARCHITECTS Co.,Ltd.

ジャストインタイムを支える活動

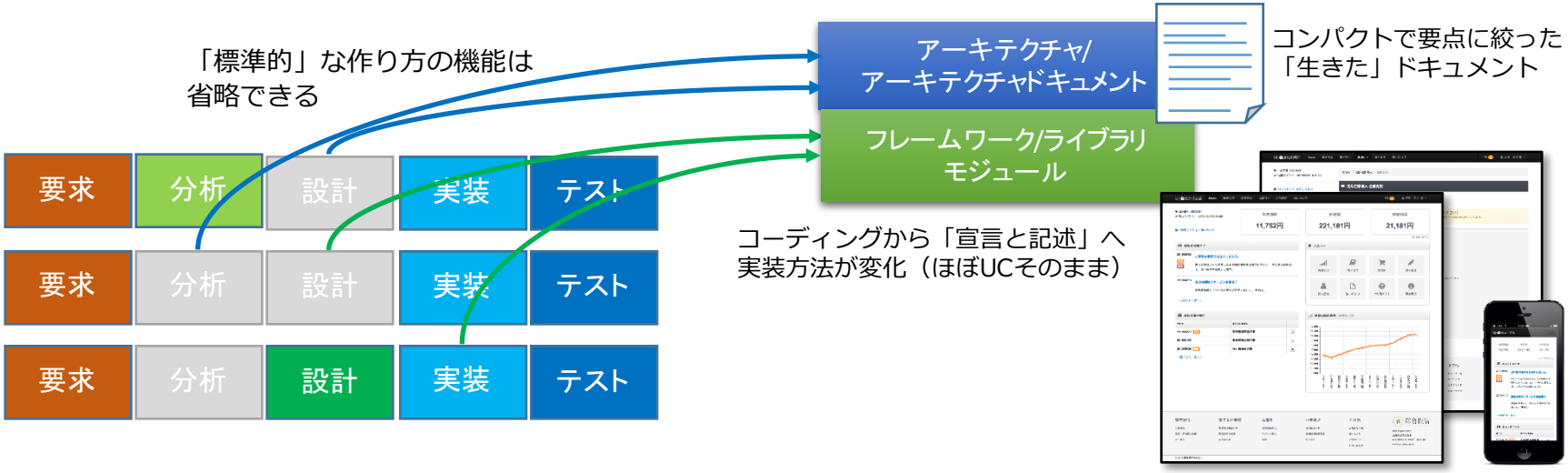


実は「保守開発」の現場で見る光景

© 2015 ZEN ARCHITECTS Co.,Ltd.

エンタープライズで「JIT」は有効か

- テクノロジーの進化で、JITベース開発での保守引き継ぎが現実的になった
  1. システムのあり方（主にUI）にiOS等を規範とするコンセンサスが醸成されてきたこともあり、技術がシンプルに洗練されてきた。（J2EE→Rails→SPA+MSAの流れ）
  2. 利用技術が絞られ、標準的な利用方法で作成できる割合が増加。カスタムコンポーネントの減少。（VB/Web/RIA → モダンWeb～フレームワーク適用範囲拡大～専用設計の減少）
  3. さらに、OSSで提供されるモジュール、ライブラリの適用範囲拡大に伴い、記述コード量の大幅な減少（同一システムでコード量が1/5程度に）
  4. コードベースが小さくなることで、ドキュメントの主目的である”引き継ぎ”をコードレベルで行う「コードの共同所有」。それに伴いドキュメントの目的が、設計方式や実装パターンの共有へ移行。実装後に清書する「ドキュメント・レイト」や、継続的修正が前提の「リビング・ドキュメント」で実用的な保守が可能になった。

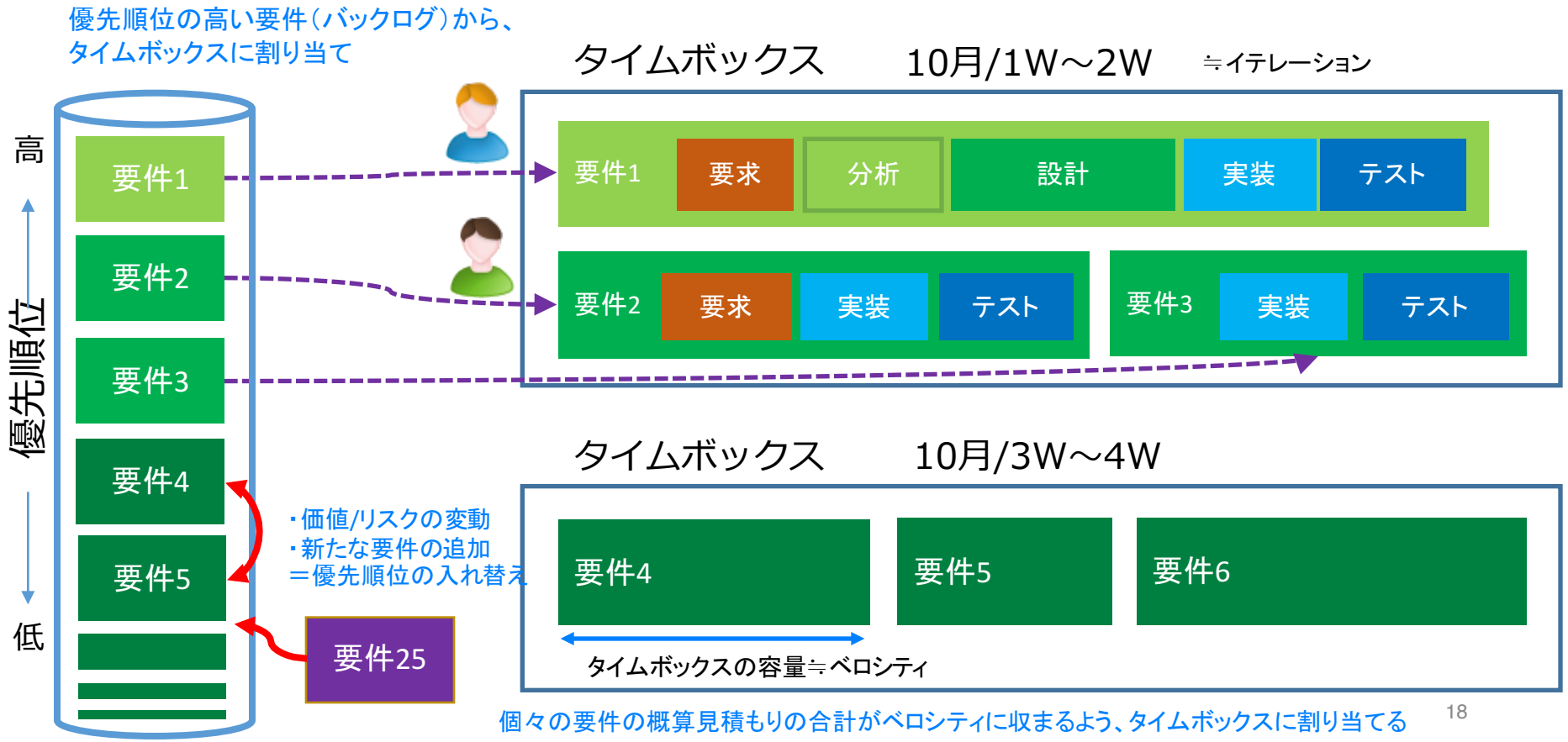


※本資料は株式会社ゼンアーキテクツによる調査と考察によるものです。

© 2015 ZEN ARCHITECTS Co.,Ltd.

適応型（価値駆動・リスク駆動）

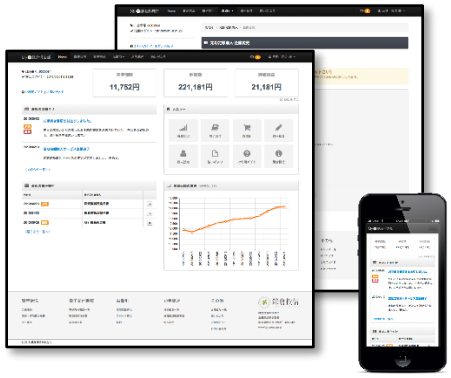
- 作業順序を決定するための要求管理方式
- プロジェクト開始時に、要件を価値/リスク基準で優先順位を設定。
- 期間 x リソースのタイムボックスに割り当てながら進める。



アジャイル"型"で何が得られる？

得られるもの

1. 不必要な作業の削減
   - JITの効果
   - ＝生産性向上＝コスト最小化（最適化）
2. リスクの減少
   - 反復型の効果
   - リスクの早期識別と早期（小さいうちに）対処
   - ＝再作業の削減 & 我慢（＝ビジネス価値低下）の最小化



- 再作業の原因（リスク）のほとんどは「要求」
  - 「ほしいものはこれじゃない」
- 動くものを触ってもらう
  - ITは、社会活動全体から見るとそこまで成熟していない（モノを見るまで安心できない）

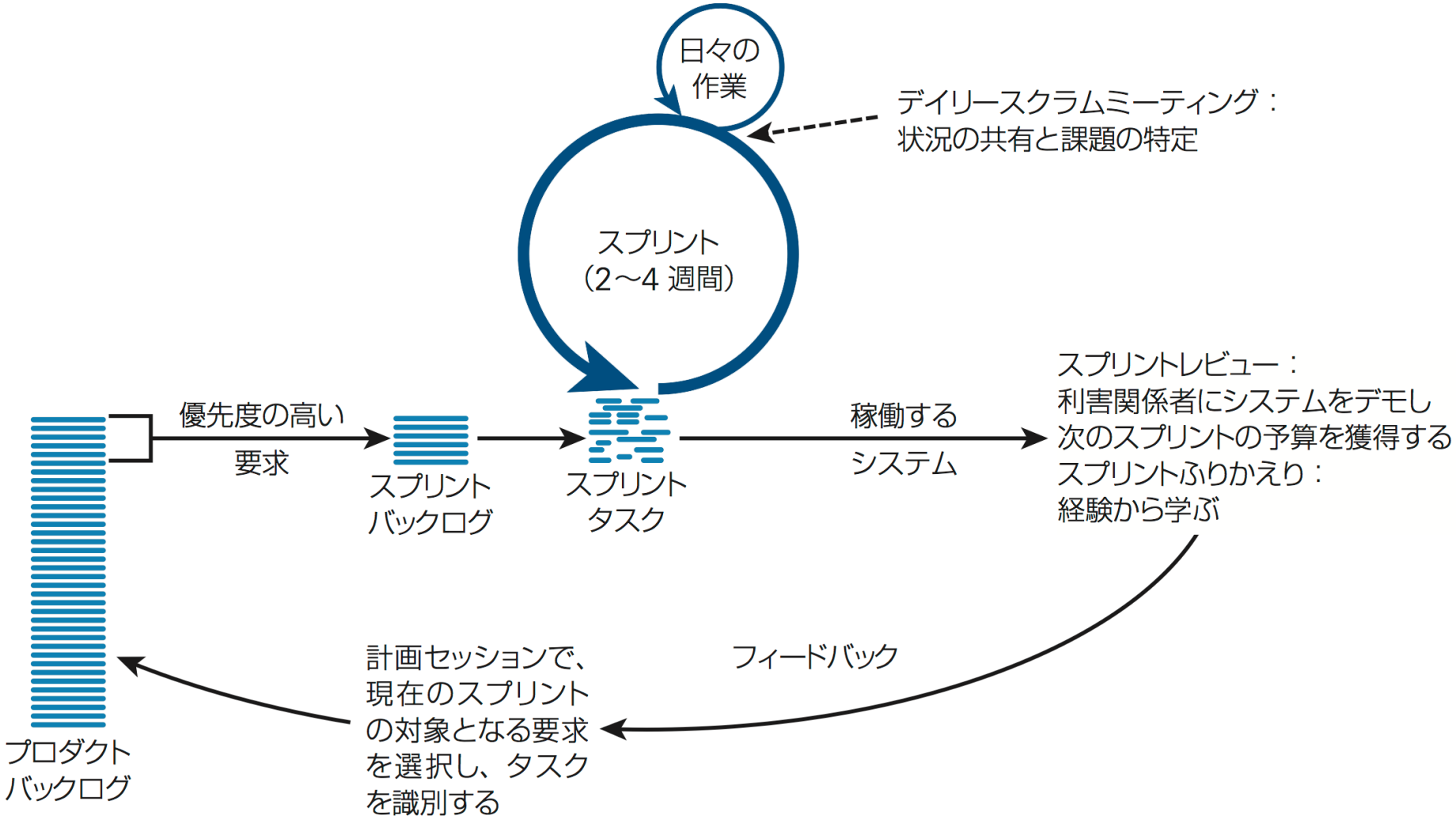
© 2015 ZEN ARCHITECTS Co.,Ltd.

スクラム(Scrum)の特徴

- プラクティス:
  - プロダクト・バックログ
  - 価値駆動ライフサイクル
  - 自己組織化
  - リリース・プランニング
  - スプリント・プランニング
  - 日次スクラムミーティング
  - スプリント・デモ
  - ふりかえり
- ロール:
  - スクラム・マスター
  - プロダクト・オーナー
  - チーム・メンバー

© 2015 ZEN ARCHITECTS Co.,Ltd.

スクラムの基本的な進め方



© 2015 ZEN ARCHITECTS Co.,Ltd.

アジャイルの期待効果３
«エンタープライズへの拡張»

不確実性

開発効率

エンタープライズ

© 2015 ZEN ARCHITECTS Co.,Ltd.

「エンタープライズ」のコンテキスト

- エンタープライズ開発 ≒ 業務システム開発
  - 価値観の異なる他の組織体と連携しながら進める開発
    - 連携が避けられない、とも言う
- 計画に基づく活動
  - 事業から見るとマイルストーンが重要。人、モノ、カネを動かすタイミング。予算、期間、リソース制約も含む。
- 戦略に則った選択
  - 事業戦略、技術戦略、既存資産の利用（人、設備、組織の経験）
- 利害関係者との調整
  - 複数のユーザー部門、他のシステムの運用・保守・開発チーム、システムオーナー部門長、役員
- コンプライアンスとガバナンス
  - 業界の法律・規制、組織の規則・ルール・標準
- 規模
  - 個別システム、関連システム、企業全体

アジャイルであっても
避けられない

© 2015 ZEN ARCHITECTS Co.,Ltd.

エンタープライズアジャイル

エンタープライズのコンテキストで、アジャイル型を実践するための”拡張版“

- ディシプリンド・アジャイル・デリバリー（DAD）

  《拡張点》

  - 新規開発：ビジョンの合意、方向付けフェーズ、初期のバックログ構築
  - アーキテクチャ：アーキテクチャ、AO（アーキテクチャオーナー）、テクニカルストーリー
  - コストとスケジュール：リリース計画、レンジド・バーンダウン
  - ガバナンス：リスクリスト、リスク・価値駆動、マイルストーンレビュー

- スケールド・アジャイル・フレームワーク（SAFe）

  《拡張点》

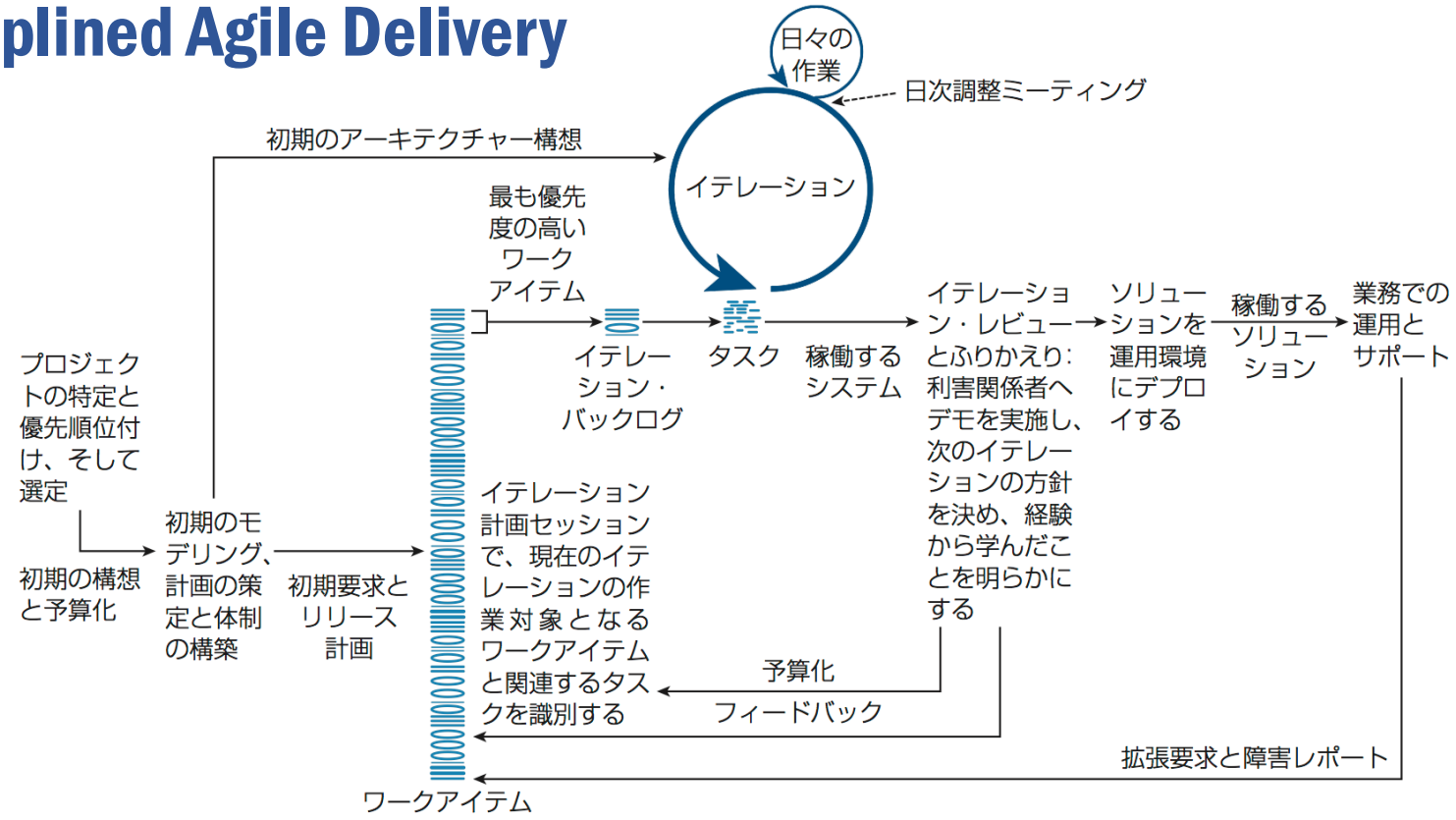
  - プログラムマネジメント：プログラムバックログ、プロダクトマネジメント
  - リリースの統制：リリース計画ミーティング、ART（アジャイルリリーストレイン）、リリースエンジニア、IPスプリント
  - アーキテクチャ：システムアーキテクト、アーキテクチャ滑走路、テクニカルストーリー
  - 企業全体：ビジネスエピック、エピックオーナー、予算と投資判断、プログラムポートフォリオマネジメント、エンタープライズアーキテクト

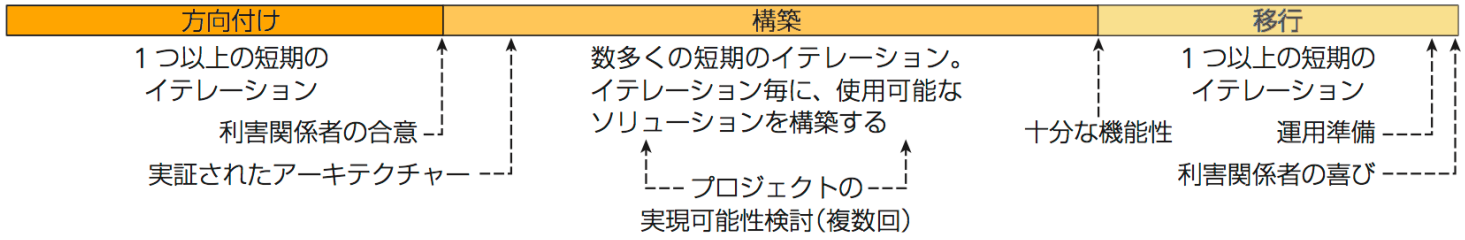
- Scrum of Scrums
- Enterprise Scrum

© 2015 ZEN ARCHITECTS Co.,Ltd.

ディシプリンド・アジャイル・デリバリー（DAD）

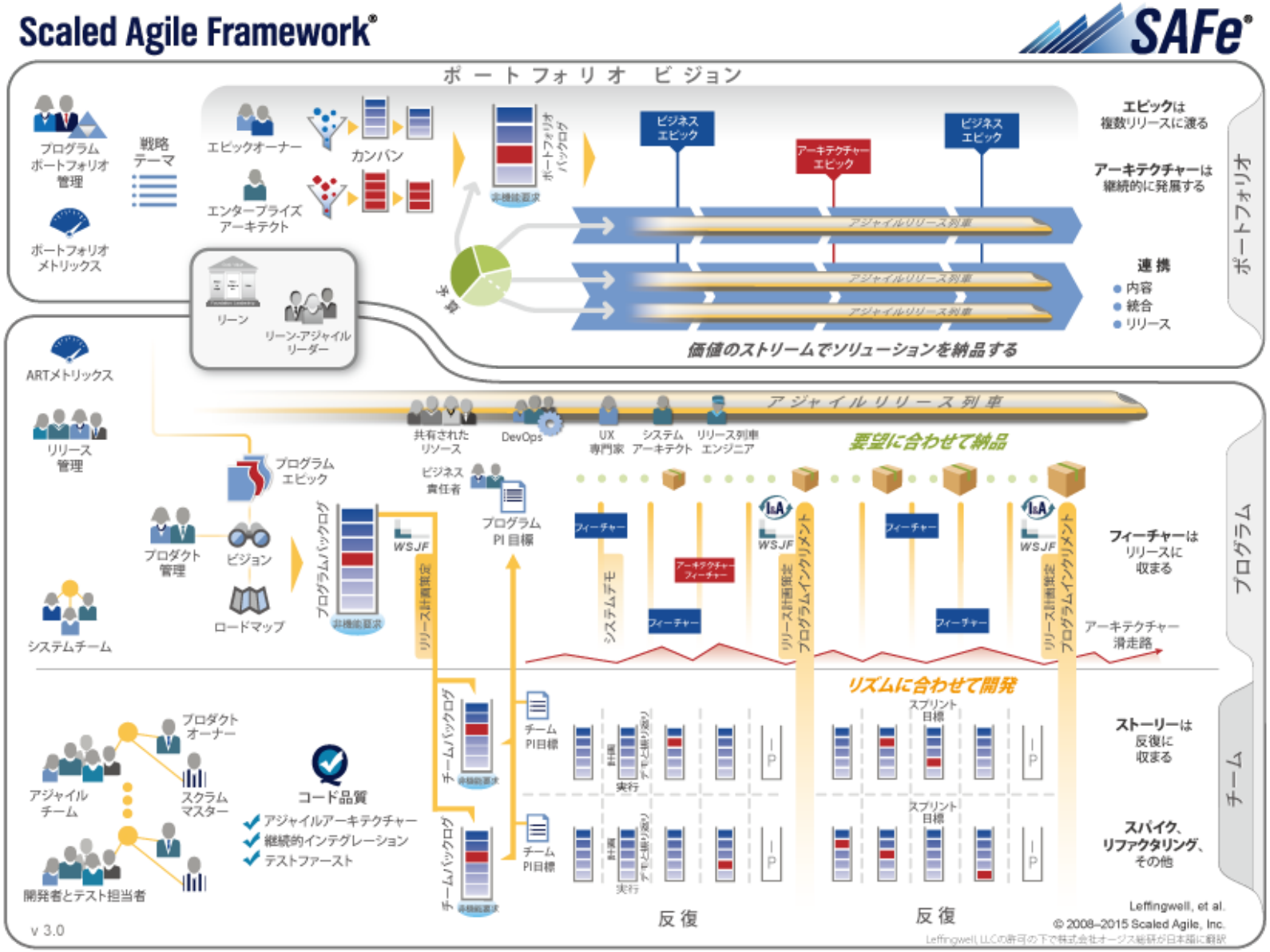
Disciplined Agile Delivery





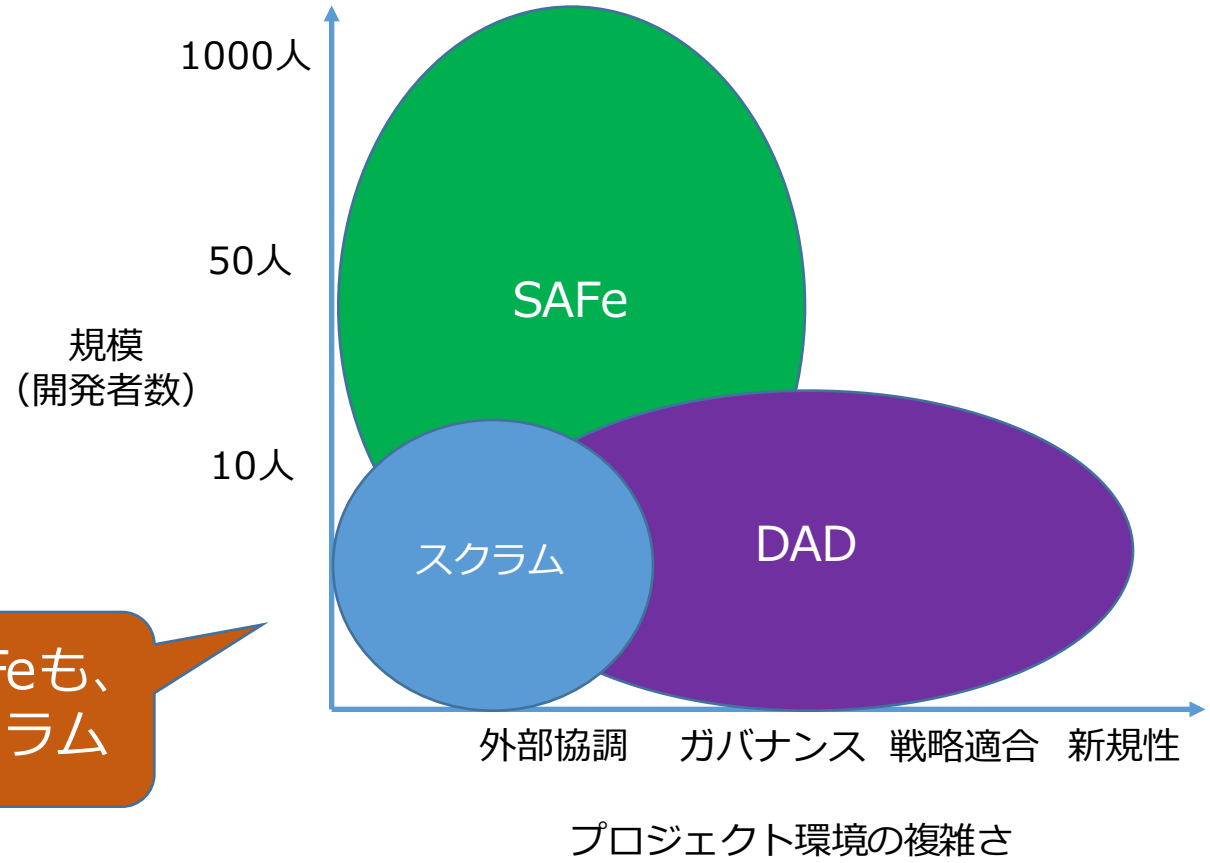
引用：ディシプリンド・アジャイル・デリバリー(2013/翔泳社)

スケールド・アジャイル・フレームワーク（SAFe）



「エンタープライズ」の守備範囲

- 「エンタープライズ」で必要となる2つの軸
  - プロジェクト環境の複雑さ
  - 規模（開発者数）



DADもSAFeも、
中心はスクラム

© 2015 ZEN ARCHITECTS Co.,Ltd.

企業へのアジャイル導入に必要な3つの要因

- 動機が適切であれば、アジャイル型の導入を検討する価値はある。
  アジャイル型の導入の成否に大きく影響する3つの要因がある。

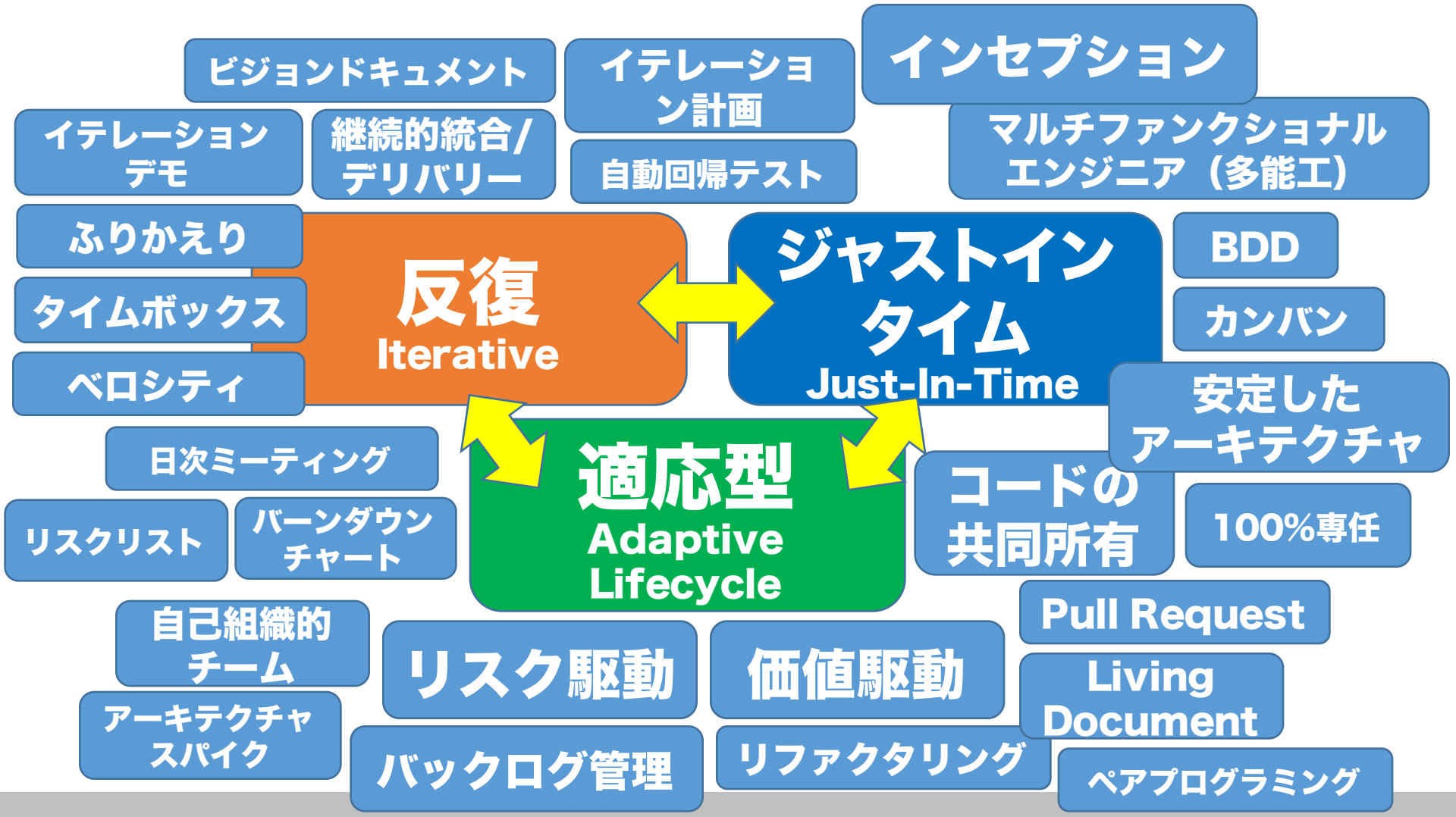
- **「人を礎とする」**
  - 「人は容易に調達可能なリソース”ではない“」ことが前提。
  - 自ら考え行動できる人（自律的に行動できる人）だけで編成できるか。経験可否は問わない。
  - PO、AO、TL ＋ デベロッパ 3～5名。自己組織的チームが目標。デベロッパは100%専任。

- **「環境とアーキテクチャを先行させる」**
  - バックログ、カンバンボード、ドキュメント共有、ソース管理の「プロジェクト推進環境」とともに、「アーキテクチャ」を先に走らせる。（PO、AO、TLで先行）
  - デベロッパー参画時には、プロジェクト推進環境とともに技術的基盤や実装方式が「ある程度」整っている必要がある。
  - 政治的調整が必要な場合は、準備するためのインセプション（方向付け）を実施。
  - クラウドや軽量フレームワークなど、Just-In-Timeで調達可能かつ低負荷で利用可能な技術スタックが望ましい。

- **「オーナーとチームの固い信頼関係」**
  - チームはベストを尽くし、オーナーはそれを信じる。オーナーとチームとの相互信頼関係が基盤。
  - 当初は見積誤差が大きい。パフォーマンスの測定と見積基礎値の蓄積、適切なイテレーション計画に向けて継続的改善。インセプションで基礎値を見出しておく。
  - 安定したアーキテクチャの元でのコードの共同所有は、パフォーマンス底上げに効果大。

© 2015 ZEN ARCHITECTS Co.,Ltd.

アジャイルは、意外にうまく設計されている

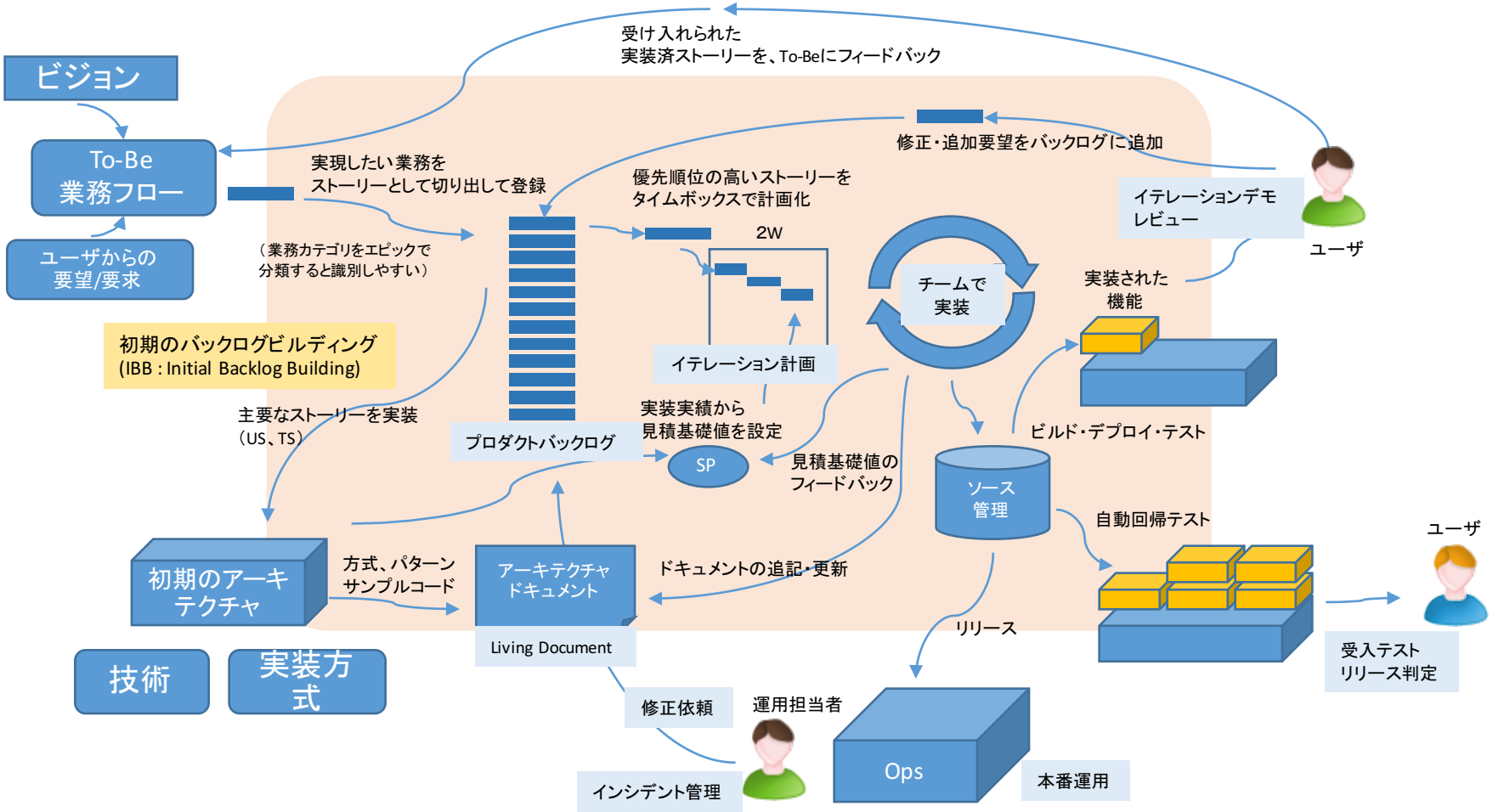


ZENARCHITECTS

※ゼンアーキテクツがお客さまの現場で実践している主要なプラクティスを表したものです

© 2015 ZEN ARCHITECTS Co.,Ltd.

企業システムとアジャイルの親和性



ZENARCHITECTS

© 2015 ZEN ARCHITECTS Co.,Ltd.

アジャイルは、予測型と「組み合わせて」使う

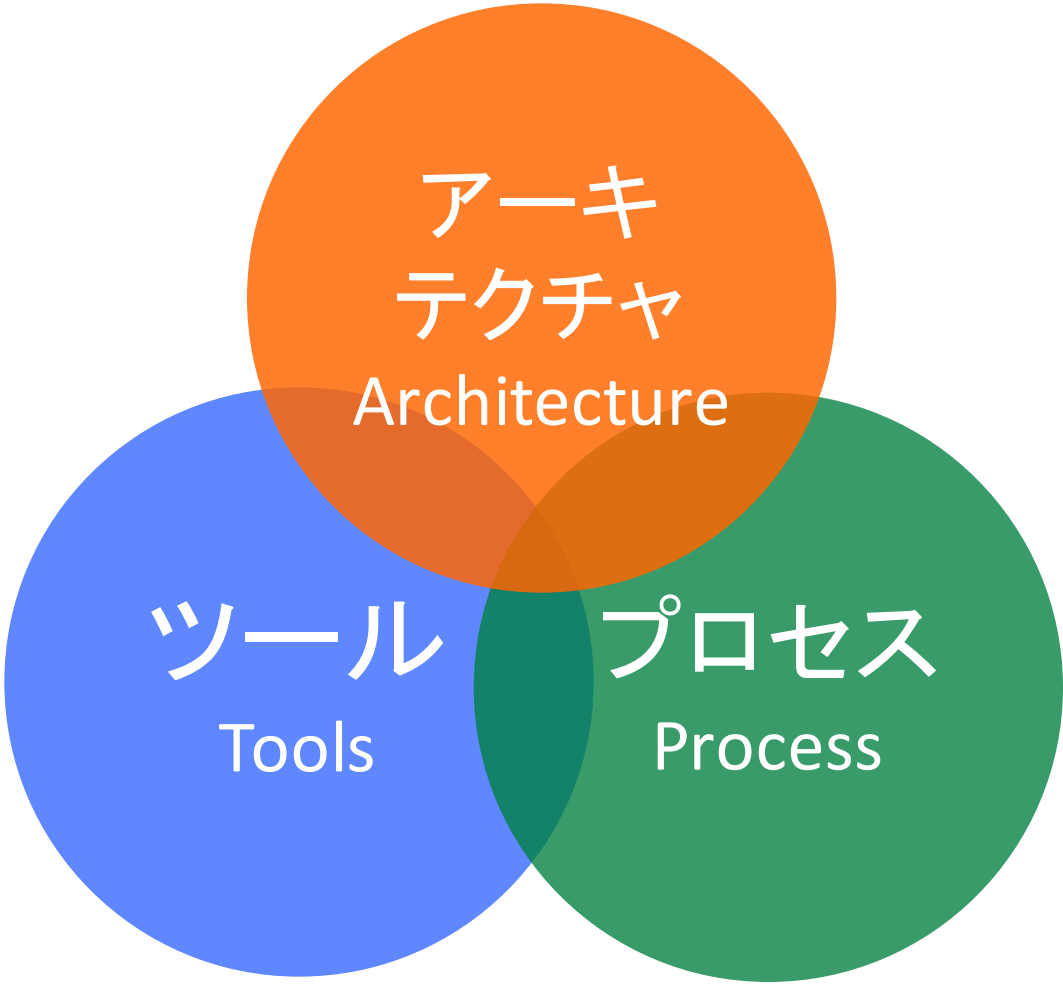
- **フロント系システム　＋　バック系システム**
  - 組織のシステム単位で組み合わせる
- **フロントエンド(SPA) ＋ バックエンド(MSA)**
  - 単一システムのレイヤー単位で組み合わせる
- **インセプション(設計) + プロダクション(製品化)**
  - 単一システムの工程で組み合わせる

「全てをアジャイルで」という発想は捨てる。

© 2015 ZEN ARCHITECTS Co.,Ltd.

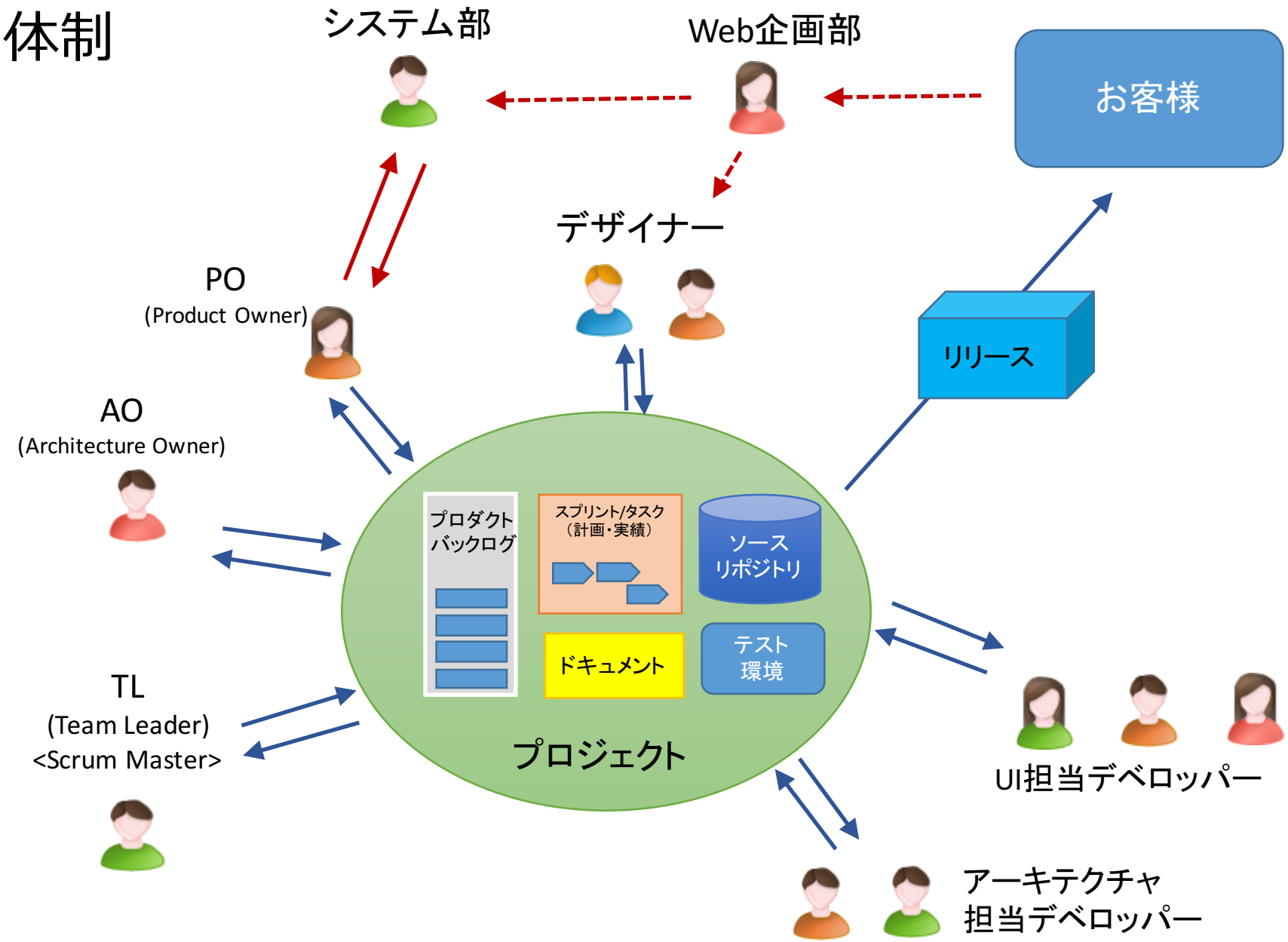
金融機関中規模チームでのDAD適用例

- ツール
  - JIRA http://www.atlassian.com/
  - Confluence
  - Bitbucket
  - Jenkins
  - SWAT http://www.smartekworks.com/
- アーキテクチャ
  - SPA/HTML5/knockout.js
  - MSA/PHP/Laravel/MySQL
  - RHEL
- プロセス
  - Disciplined Agile Delivery



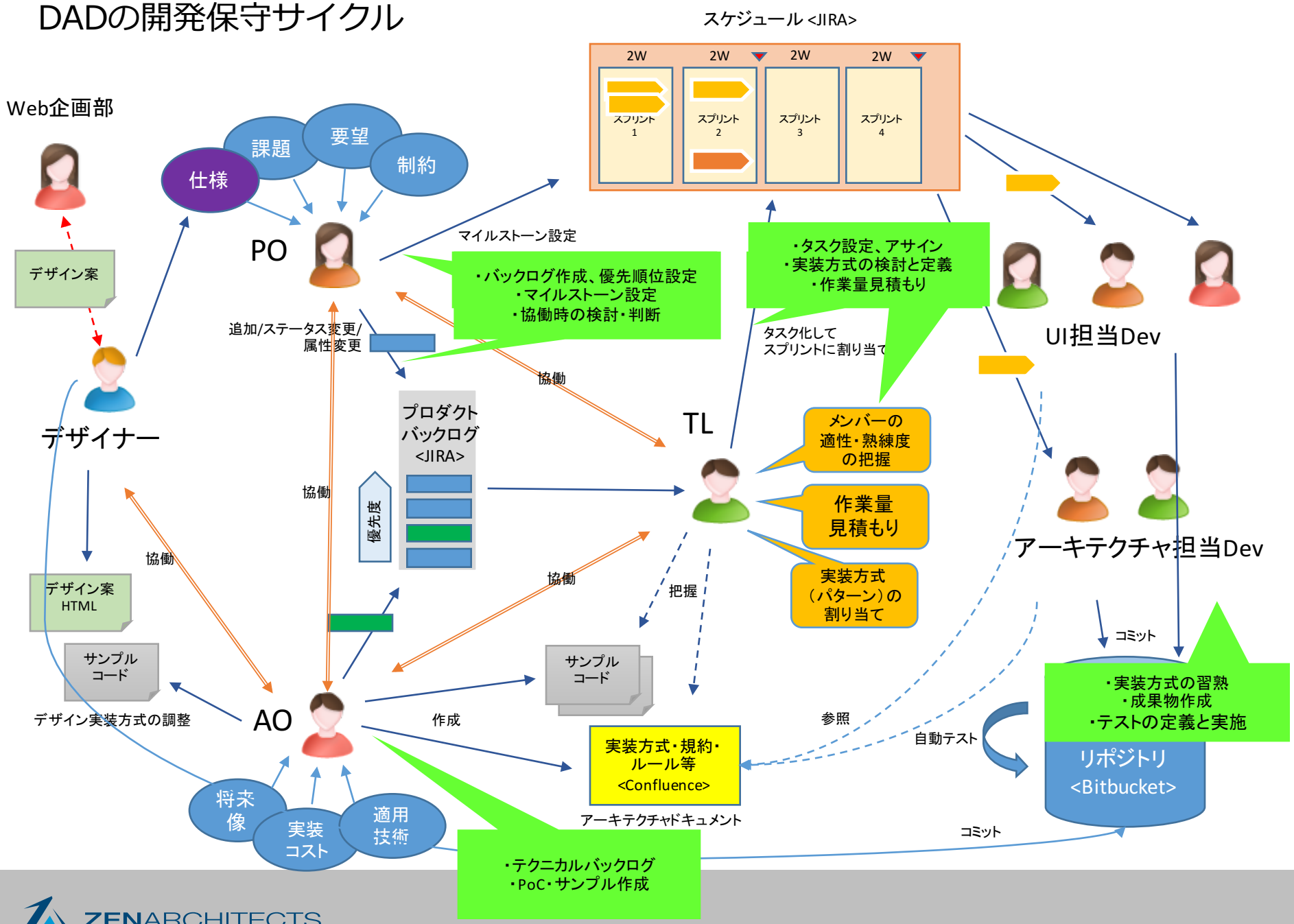
© 2015 ZEN ARCHITECTS Co.,Ltd.

体制



© 2015 ZEN ARCHITECTS Co.,Ltd.

DADの開発保守サイクル



ZENARCHITECTS

© 2015 ZEN ARCHITECTS Co.,Ltd.

アジャイルは不確実性に立ち向かうための「道具」である

1. **未来が想定できるかどうか。不確実性を認識する**
   - 想定できるなら「予測型」で計画駆動が前提。できない計画は立てない、そこに不確実性がある
2. **“事業者”と“システム提供者”の不確実性は、特性が異なる**
   - システム開発のリスクは「要求」「技術」「スキル」
3. **アジャイルは、不確実性と開発効率に対するソリューション**
   - 各施策やプラクティスは、ソフトウェアエンジニアリングの歴史の延長線上にある
4. **エンタープライズアジャイルは、単にアジャイルの適用範囲を広げたもの**
   - そのぶん、シンプルさは損なわれている
5. **”アジャイル適用時の不確実性”に立ち向かうためのソリューション**
   - システム提供者が直面する「新規性」「見積り」「規模拡大」をまとめて「エンタープライズ」
6. **アジャイルは、思ったよりもうまく設計されている**
   - 単一のプラクティス適用よりも、まとめて適用した方がうまくいく
7. **アジャイルは、組織が手にすることのできる新たな「道具」に過ぎない**
   - 適切な局面で、適切に組み合わせて利用できれば良い。道具として使えるようになっておく

© 2015 ZEN ARCHITECTS Co.,Ltd.

zenarchitects.co.jp

facebook.com/zenarchitects

© 2015 ZEN ARCHITECTS Co.,Ltd.