派生開発推進協議会
AFFORDD

派生開発カンファレンス2024
ワークショップ
派生開発とアジャイルの「いいとこどり」

～派生開発とアジャイルのアプローチを徹底比較し学ぼう～

サイボウズ株式会社 永田 敦
派生開発推進協議会 T4/T6研究会メンバー

2024年5月24日　13:25~15:25

主催者紹介

派生開発推進協議会
AFFORDD

T04研究会 「XDDP」とテストプロセスとの接続
堀川透陽

T06研究会 Agile開発との連携
永田 敦
斎藤賢一
葛西孝弘
工藤寛
中村勝志
星野充史
本田英稔

ワークショップ概要

派生開発推進協議会
AFFORDD

- アジャイルも派生開発も、既存の開発資産に変更や機能追加を行い、価値あるソフトウェアを提供することを目指しています
- XDDPの勉強会資料では、アジャイル開発も派生開発である、と述べています
- しかし、それらの開発アプローチは違います
- 本ワークショップでは、派生開発の解決したい問題と課題に立ち返り、XDDPとアジャイルがそれをどのように課題を解決しているのか、そのアプローチを対比していきます
- そこから、アジャイルがXDDPから学ぶこと、XDDPがアジャイルに学ぶことの気づきを、参加される方と一緒に探索していきます
- これらの気づきは、アジャイル開発と派生開発のさらなる工夫や進化をもたらすことを期待しています

アジャイル開発とXDDP

派生開発推進協議会
AFFORDD

- 一見すると正反対のアプローチに見える、アジャイル開発とXDDPですが、互いに学ぶべきところがあると思います。
- T6研究会は、アジャイル開発とXDDPは融合できないか、考え方、価値観を互いに取り入れることはできないか、研究してきました。
- 今回のワークショップは、XDDPの勉強会の資料を基に、XDDPから学ぶこと、アジャイルから学ぶことをあげて、いいとこどりの考えを議論しようと思います。

ワークショップのスコープ

派生開発推進協議会
AFFORDD

- XDDPのお話のスコープは広く、とても2時間では議論しきれません。そこで、今回は特に　コードの実装のあたりをフォーカスします。

- XDDPでは、3点セット（変更要求仕様書、TM,変更設計書）を書き、レビューが終わるまで、コードに手を付けません。
- 一方でアジャイル開発は、要求ごとに設計、実装、テスト、結合をイテレーティブにおこなう開発スタイルです。
- この最も違いのあるところを中心に議論をしていきます

開発の対象

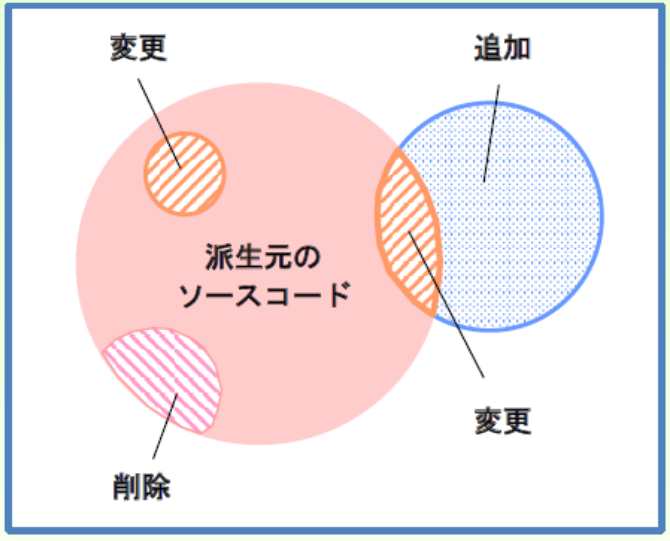
派生開発推進協議会
AFFORDD

- 違いを直接議論する前に、開発の対象になるものを確認していきましょう

1.1　派生開発とは

派生開発とは：

機能の追加、仕様の変更・削除、メンテナンスにより新たなシステムやソフトウェアを作ること



出典：古畑慶次, XDDPで現場を変える!, CASEフェスタ2010

一般的な派生開発の姿

ベースとなる既存のソフトウェアに対して

- いくつかの新しい機能が追加され
- 既存機能のいくつかの箇所で使いやすくするための変更や削除が行われ
- 前回のテストで発見しきれなかったバグや、対応を持ち越したバグの何件かに対応する

- ■ 組み込み系ソフトウェア開発のほとんどが「派生開発」
- ■ 90年代以降、ソフトウェア規模の増大、納期短縮、コスト削減などの市場要求変化にともない、パッケージソフトやエンタープライズ系ソフトにおいても「派生開発」は増加
- ■ アジャイル開発におけるスクラムのスプリントも派生開発の繰り返し

そういう定義でしたら、アジャイル開発は、ほぼすべて派生開発といえます。

アジャイル開発は、コードの変更は必然的なものととらえています

> 要求の変更はたとえ開発の後期であっても歓迎します。
> 変化を味方につけることによって、お客様の競争力を引き上げます。
>
> アジャイル宣言：アジャイルソフトウェアの12の原則から

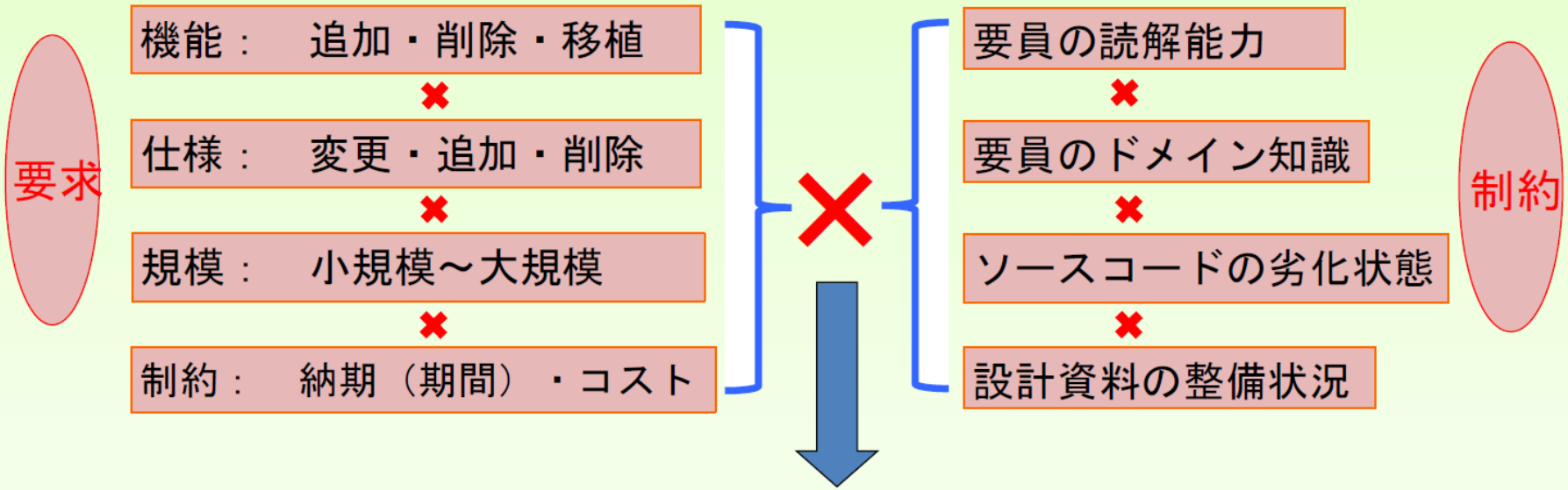
変更の理由

1. 要求の追加
2. バグの修正
3. 設計の改善
4. リソース利用の最適化

レガシーコード改善ガイド

1.2　作業内容は多様で複雑 1/2

- 派生開発では要求がきわめて多様で、実現する体制も十分ではない



多様なプロジェクト
- わずか1000行の変更だけの１人プロジェクト
- 10万行の機能追加と変更が混じった10人のプロジェクト

派生開発では，これらの多様性にうまく対応することが必要

1.3 「部分理解」という制約の中での作業

- 反省会での弁明・・・「全体を理解できていなかったから」

- 現状
  - 設計書・・・・・・・・理解の助けにならない
  - ソースコード・・・保守性無視 ＋ 劣化の進行
  - 担当者・・・・・・・・ソースコードの読解技術の不足

→ 全体を理解できる状況ではない

※「全体を理解すれば問題は解決する」と思っているかぎり、
理解できなかったときの対応が想定さることはない

派生開発では「部分理解」の制約の中で変更作業が強いられる

完全理解はできるのか？

派生開発推進協議会
AFFORDD

- アジャイル開発も部分理解になる
- すべてのコードを理解しようとすると、かなりの認知負荷がかかる

そもそも、全体を詳細に理解することは難しい

- スコープを小さくする
  - 問題空間　：　要求を小さく分解する
  - 解決空間　：　仮説検証を小さく回す
- チームで対応する
  - 多様性の利用

1.4　変更量によっては「一人プロジェクト」に

- 変更規模が小さい場合

  一人の担当者の中に“すべてが隠された状態”でソースコードが変更される「一人プロジェクト」になりやすい

- 変更規模に関わらず

  「一人プロジェクト」のスタイルに慣れている人が集まっていると、各担当者の変更仕様情報が公開されず、お互いの担当の中で「一人プロジェクト」の状態に陥る
  （疑似一人プロジェクト）

派生開発では、「一人プロジェクト」が発生しやすい

一人プロジェクトの課題

派生開発推進協議会
AFFORDD

一人プロジェクト　＝　ほかの人とコミュニケーションをあまりとらない

暗黙知になりやすい
　　一応、フォームにのっとりドキュメントは書くけれど
　暗黙知化しているので、あまり書かない

派生開発の要求はスコープが小さく見えるので、
問題空間の理解もそこそこに、すぐに解決空間に走り
コードをいじる

アジャイルでも一人プロジェクトになってしまうことがある

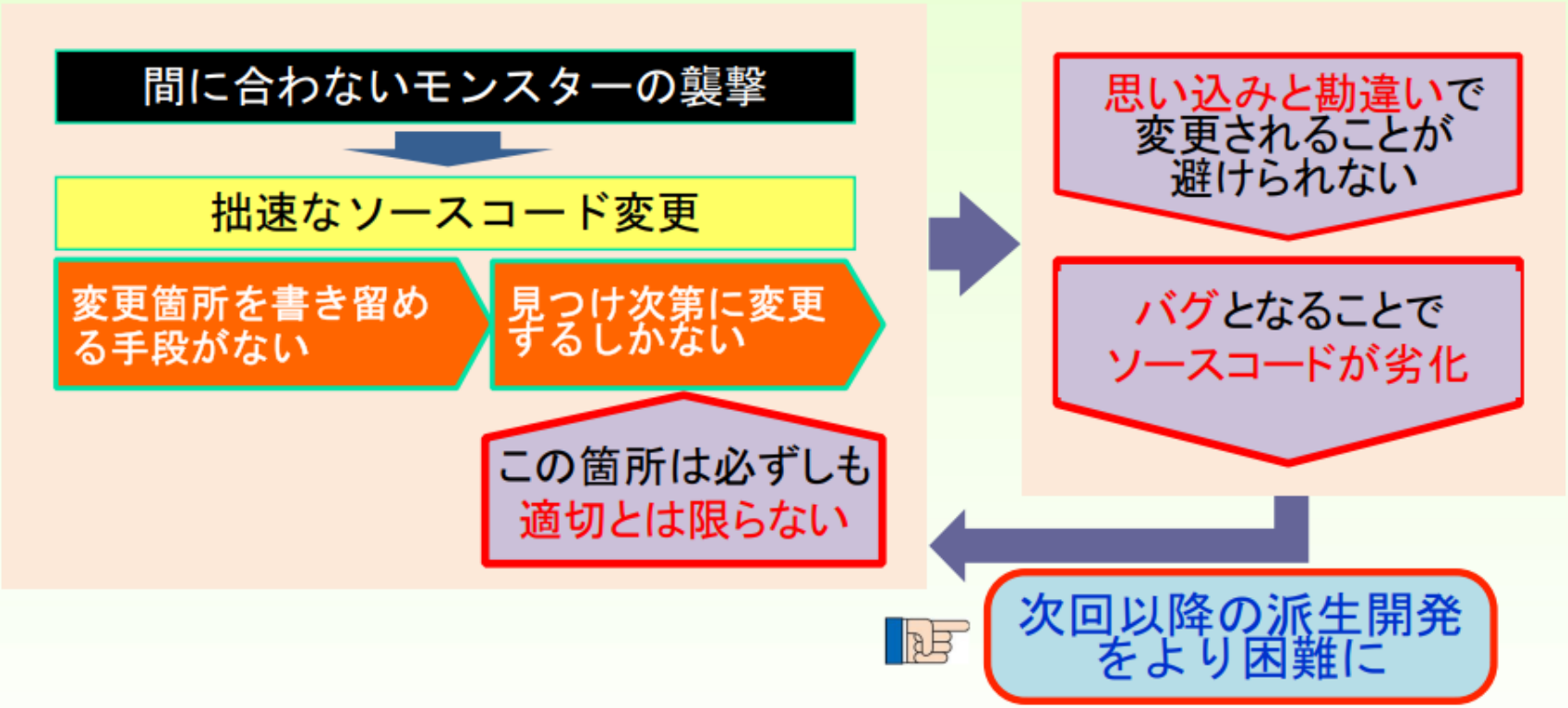
仕様の誤り、漏れが起こる

見積もりが悪い（なかなか終わらない）

品質が悪い

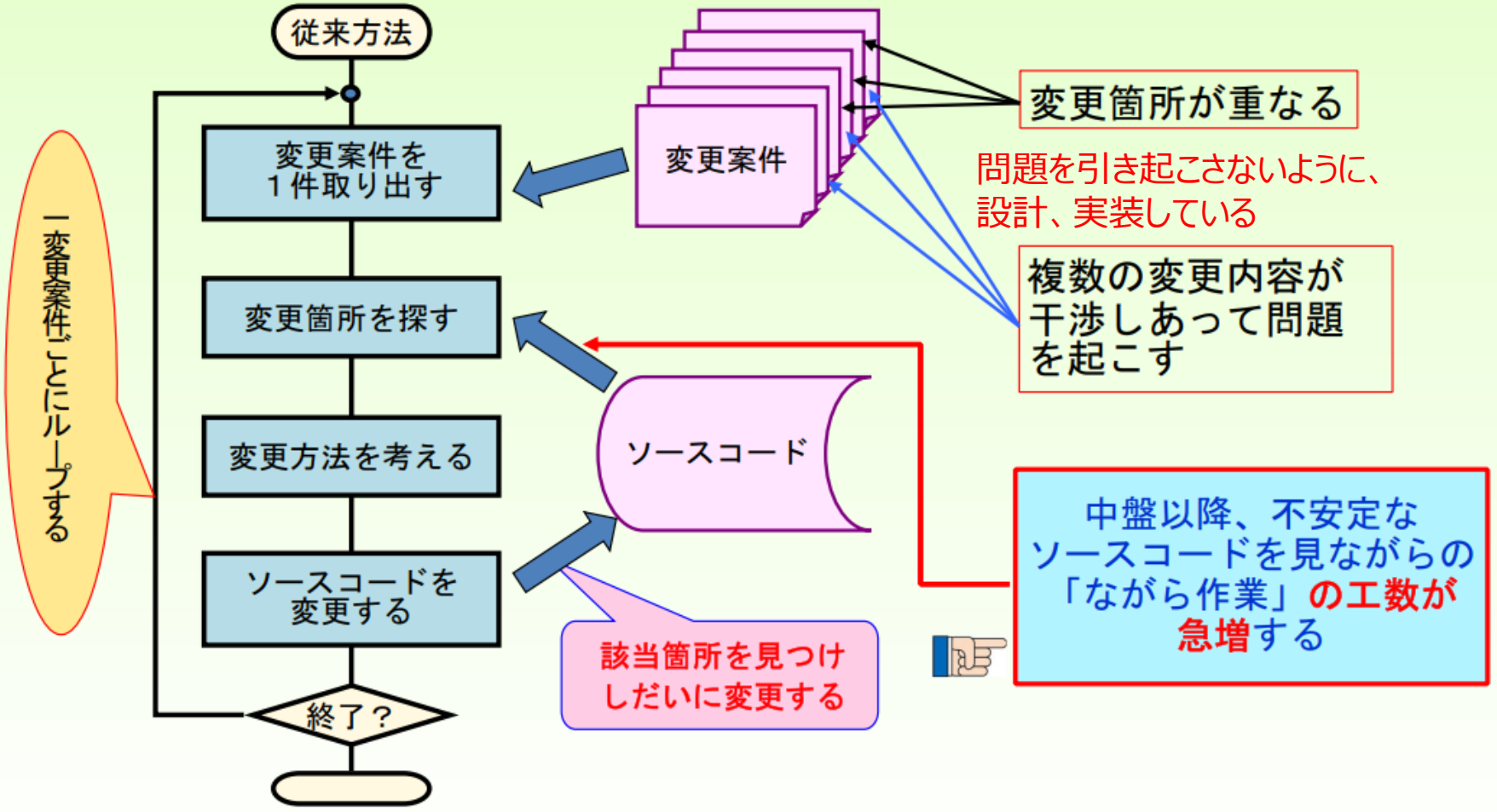
ソースコード劣化の悪循環が派生開発をより困難に

- 変更箇所を書き留める手段がない多くの派生開発状況では、「間に合わないモンスター」の襲撃に伴う拙速かつ不確実なソースコード変更が、派生開発をより困難にする悪循環を生む。



2.2　拙速なソースコード変更　2/2

- 変更箇所は見つけ次第ソースコードまで変更



アジャイルでもあるあるパターン

アジャイルのプラクティス（活動）の意味を理解せず、
形だけをまねたもののばあい、
これと同じようになることがある。

要求（バックログ）のレビューもなく品質が悪いと、
仕様も生煮えのまま
コードをすぐにいじってしまうことになる。

振る舞いの漏れ　（エラー処理やコーナーケース）

コードレビューやテストも的確でないと、
内部品質が悪化し、
デグレードやバグを生む。ネガティブスパイラル

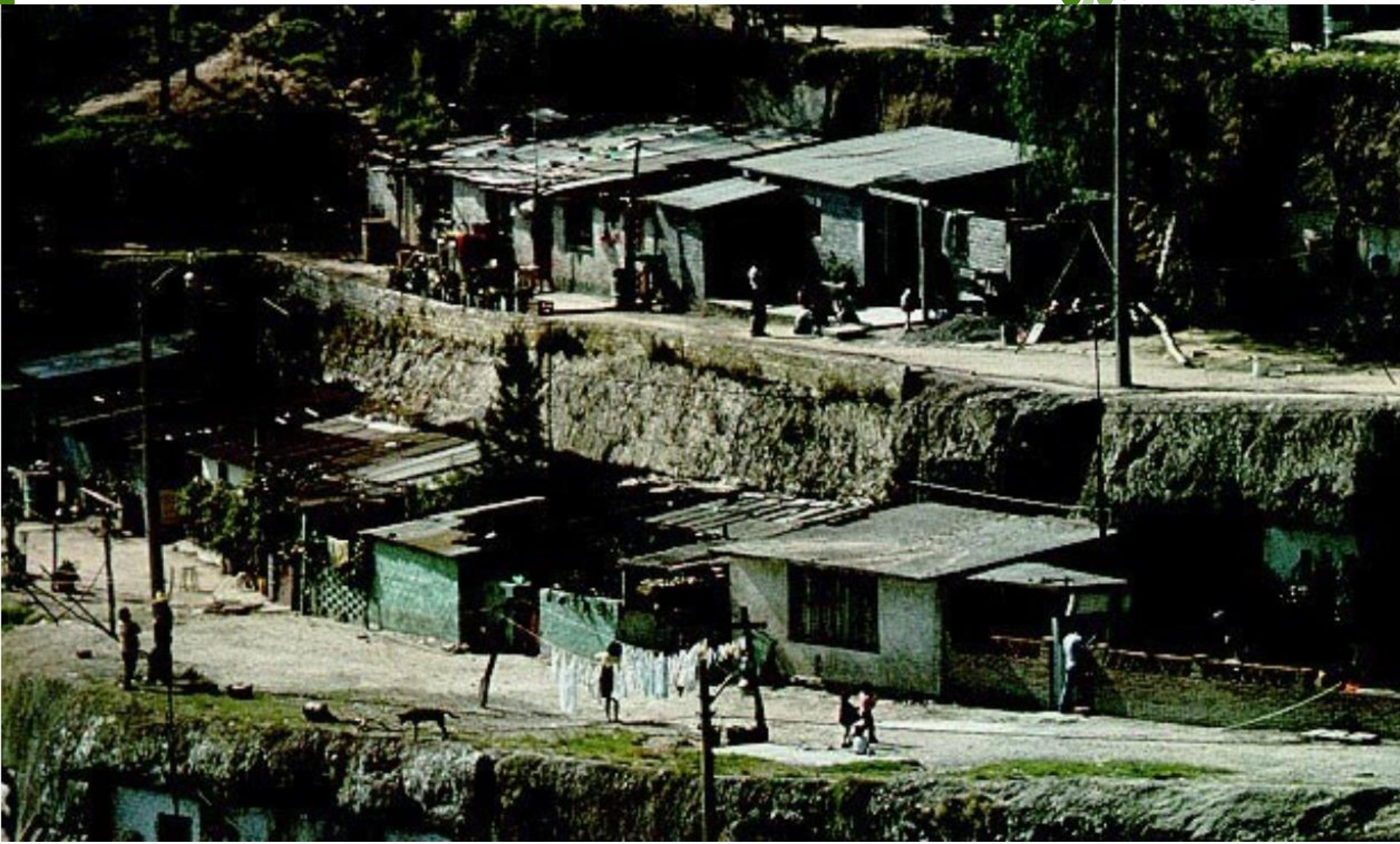
アジャイル開発での失敗の原因

巨大な泥団子

派生開発推進協議会
AFFORDD

シャンティタウン

派生開発推進協議会
AFFORDD



派生開発推進協議会
AFFORDD

取り巻く環境、制約条件はアジャイル開発も変わりません

1.5 新規開発とは異なる不具合発生のパターン

- 派生開発と新規開発における不具合発生パターン

種類		バグのパターン	対応
新規開発		• 要求仕様で求められていることに対する不適合	• 要求仕様の精度向上 • 要求仕様との適合性レビュー • 仕様実現技術の習得
派生開発	機能追加		
	変更	• 依頼された変更の解釈の違い	• 変更要求仕様の表現の工夫
		• ベースのソースコード（旧仕様）に対する認識の不足で変更箇所を漏らした	• ベースを理解する技術の確保 • 関係箇所／影響箇所の発見方法の工夫

違う！

派生開発と新規開発とでは、発生する不具合の様子が異なる

派生開発におけるバグの原因

派生開発推進協議会
AFFORDD

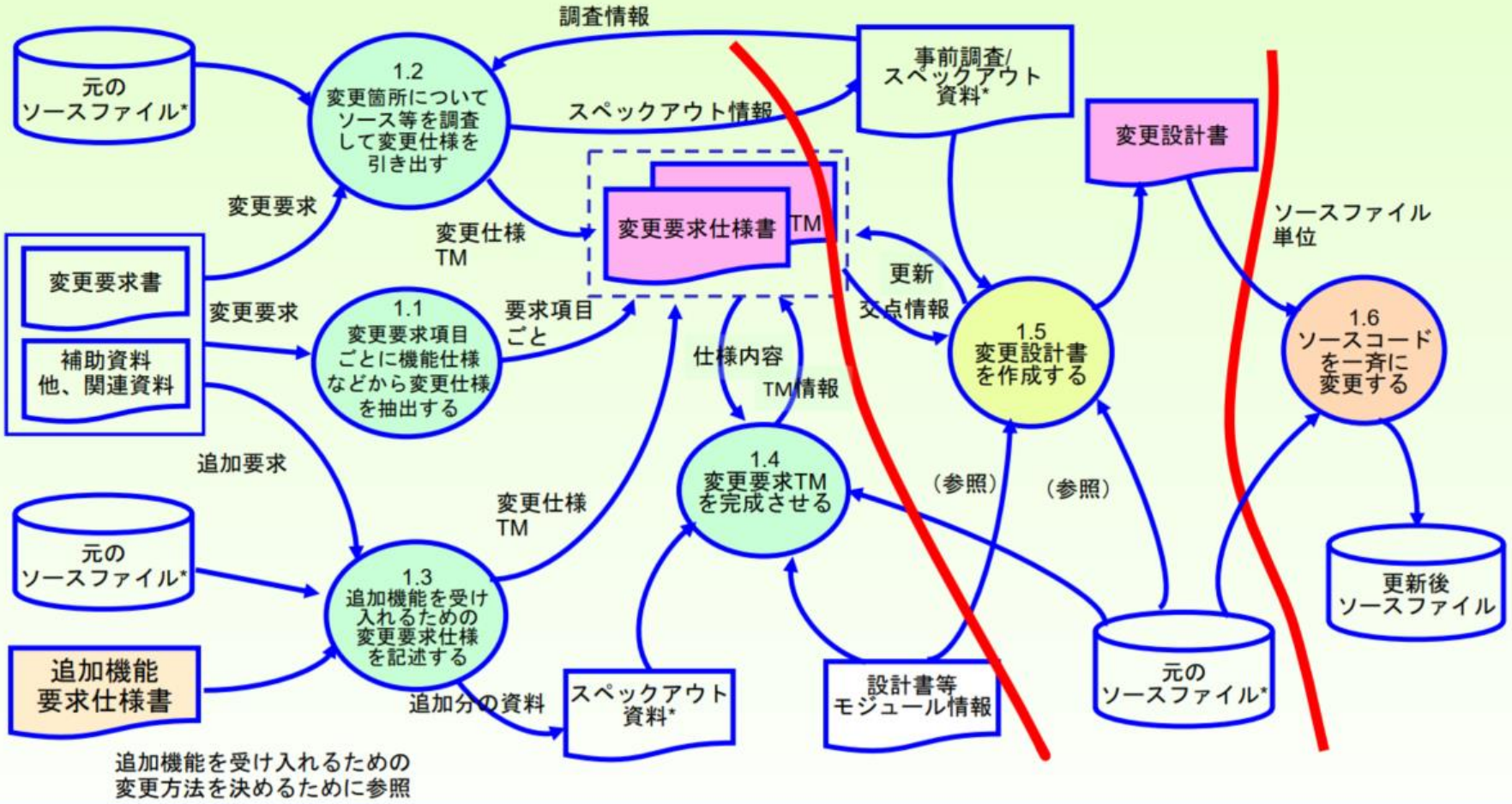
問題空間　　変更の要求仕様の品質

解決空間
- 変更箇所（影響箇所を含む）の特定
- 既存コードの理解
- 変更箇所の変更方法
- 既存コードの質

<3.2　XDDPは２種類のプロセスを並行させる>

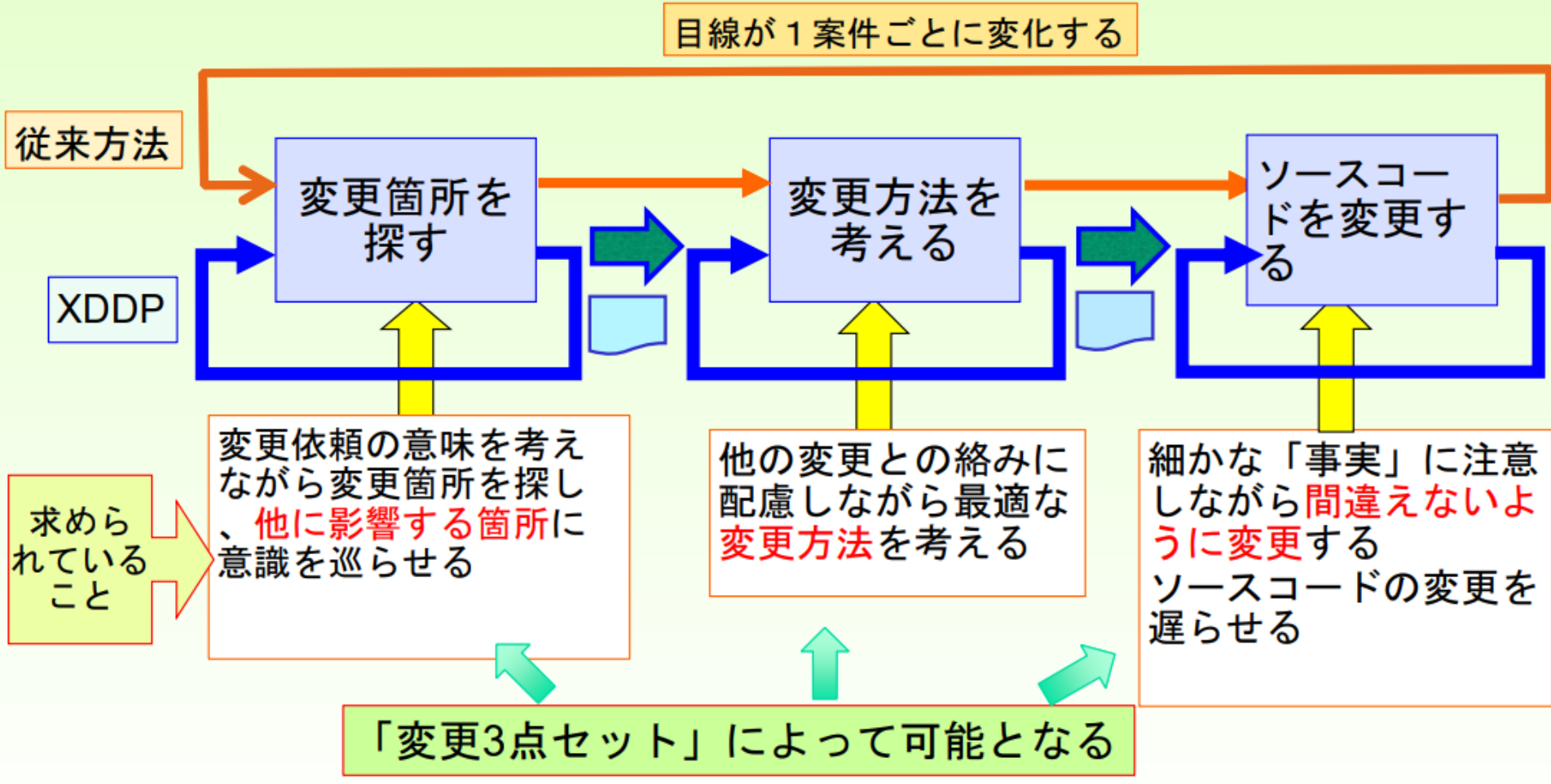
3.2.3　変更プロセス：すべての変更を扱うプロセス

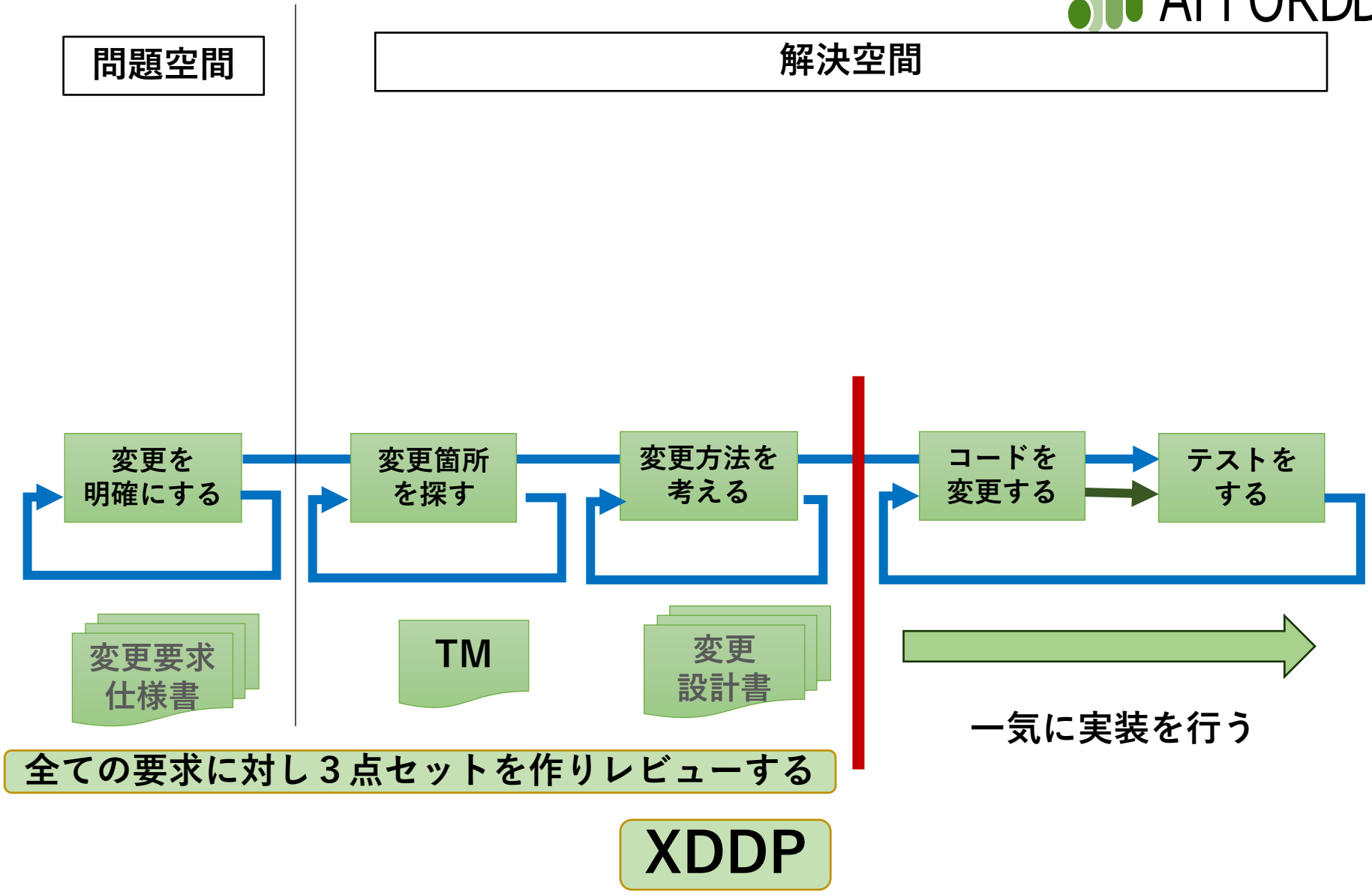
- 変更プロセスでは、「変更３点セット」の成果物に変更箇所や変更法を記述し、すべて揃った後に一斉にソースコードを変更する



3.4　XDDPは性質の異なる行為を混同させない

- 従来方法では、１件ごとに「目線」が変化する





5.4ソースコードの変更
5.4.1　一気にソースコードを変更する　1/2

- 変更設計書に基づいて一気にソースコードを変更する

ソースコードの変更を遅らせる理由 ⇒ 変更が相互に影響し合う状態になっているため、すべての変更箇所を明らかにしてからソースコードに着手することで、混乱を緩和するのが狙い

  - 抜け駆け変更を許すと、後の人に変更を押し付けることも生じる
  - ソースコードの変更に必要な工数は確認されている
  - ソースコードの変更作業の段階で人を投入することも対応可能

- 実装工程の生産性が示すこと

生産性が高い	変更設計書に必要な情報が拾いきれている
生産性が低い	変更のための情報が不足していて、変更しながら考えている この時、立ち止まって考えたことは変更設計書には記述されない

XDDP・・ソースコードの変更は「１回」で済ませることで、ソースコードの劣化を防ぐ

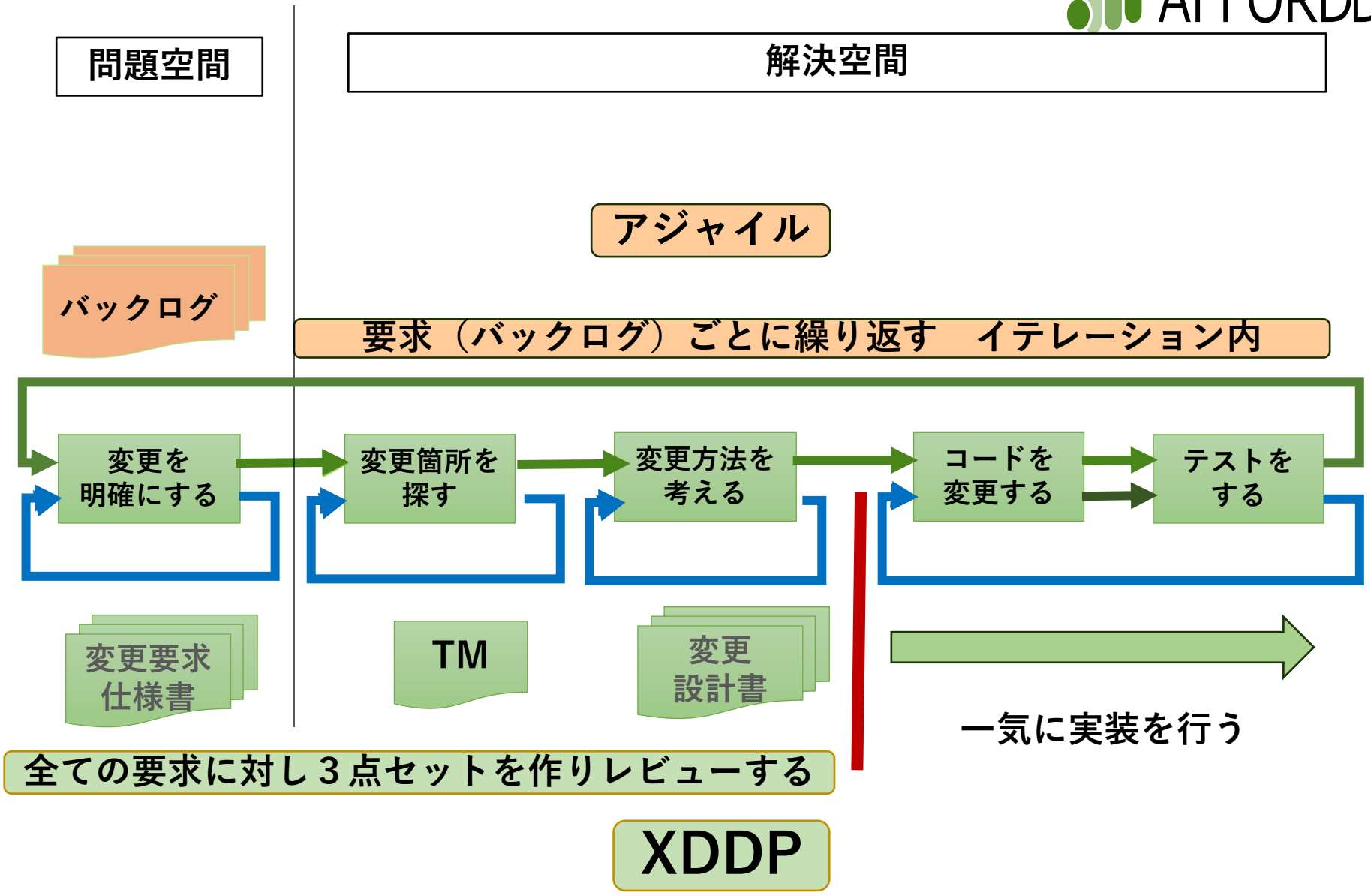
コード変更を遅らし、一気に行う

派生開発推進協議会
AFFORDD

- 最適な変更の設計をしていきたい
  - 別な要求が同じ個所を変更することになったときに、それぞれの要求に合う最適な設計をしたい
- 変更の影響による混乱を緩和したい
  - 全ての変更箇所を明らかにすることで、影響し合う状況が把握しやすい
- よりコーディング自身の効率を上げたい
  - 変更設計書に以上に挙げた情報を盛り込み設計することにより、迷いがなく一気にコーディングすることができる
- コードの劣化を最小限にしたい
  - 一気に行うことで、コードの変更の回数を最小限にできる

ディスカッション1

コーディングを遅らすメリットはなんでしょう？



アジャイルの特徴

派生開発推進協議会
AFFORDD

- アジャイルでも、問題空間と解決空間は分けようとしている
- 問題空間
  - スクラムではバックログ、Leanでもチケットで要求は表している
  - その要求における表現の質は、課題を持っているところもある。
    - USDMの考え方は勉強になる
  - 要求のプラクティスがそろっている
    - リファインメント：バックログの内容を洗練する
    - ユーザーストーリマッピング
    - 実例マッピング（BDD）
- 解決空間
  - バックログごとに、設計、実装、ビルドを行い、メインブランチに結合し、テストを行う。
    - そこで要求どうりに実装されているかを確認できる
  - “Doneの定義”という、バックログの完了の定義がある

アジャイル開発の違い

派生開発推進協議会
AFFORDD

- バックログ単位でコーディングを行う
  - PRO
    - すぐにテストをすることができる
    - 実装による問題や、仕様、設計の想定外の問題をいち早くフィードバックし学ぶことができる。
  - CON
    - コードの変更が頻繁に起こるので、コード劣化のリスクがある。
    - 同じところを何回も変更する

変更によるコードの劣化

派生開発推進協議会
AFFORDD

XDDPでは、
ソースコードの変更は「１回」で済ませる
ことで、ソースコードの劣化を防ぐ

ソースコードの変更は、数が少ないほど、ソースコードの劣化が少ない

ソースコードを変更すると、ソースコードは劣化する

派生開発推進協議会
AFFORDD

ディスカッション2

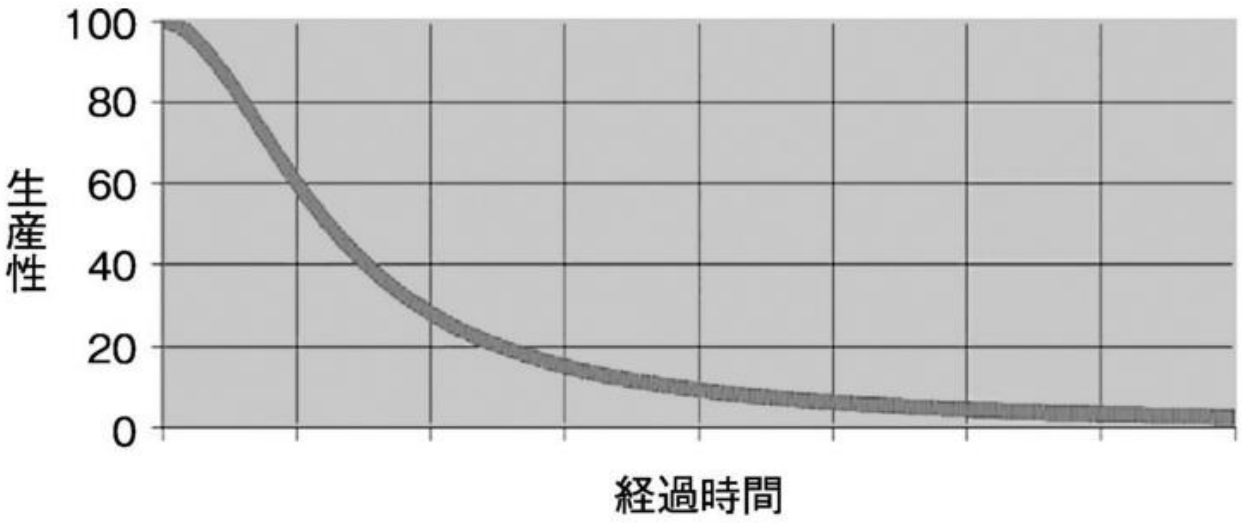
Q: なぜ、コードを変更すると、劣化してしまうのでしょうか？

Q: なぜ、コードは変更をかけると劣化してしまうのでしょうか？

変更するたびに、コードを2，3か所壊してしまう

コードを追加したり変更するには
現在のコードにある“もつれ”、“ねじれ”“結び目”をいくつも
理解しなければならない。

そして、コードを追加変更した後には、新しいもつれ、ねじれ、結び目
がいくつもコードに加えられる。そして、以下のように生産性は
時間とともに0に近づいていく



派生開発推進協議会
AFFORDD

Clean Code：　Robert Matin

可読性

コードを追加する際にも、常に既存のコードを読まなければならない。

コードを読みやすくすることは、書きやすくすることを意味する

コードは何度でも洗練しなければならない

洗練：リファクタリング
テストが必要
イテレーティブなプロセス
少しずつ、コードを変えていく

TDD
リファクタリングが入っている

派生開発推進協議会
AFFORDD

コードの変更を1回で済ませる

コードの変更は一度でよいものができるのだろうか？

”一遍のプログラムロジックであっても、それがエレガントで、プロの手による仕事とみなされるためには、3度も4度も書き直されなければならないことが多い
ヨードン”

変更設計書に“十分な“情報を書ききれることができるか？

変更設計書はコーディングの直前のところまで書くといわれる
その組織でわかるところまで書く：目的とスキル

その“直前のところまで書いたもの”は、最適なものなのだろうか？

なので、レビューする

それでも不確実なもの、
（コーディングをし、テストをして失敗すること）
はないだろうか？
その失敗は学びになる

実験（仮説検証）と、熟考やレビューとのバランスをとること

アジャイルから見たXDDP

派生開発推進協議会
AFFORDD

- 要求フェーズでの、要求の表し方は参考になる
  - アジャイルにおける要求と仕様
    - 要求はバックログのユーザーストリーとして対応している。
    - 仕様はあまりドキュメントとしては書かれていないことがあり、本当にあるべき振る舞いが分からなくなることもある
    - 負担のない仕様の扱いが課題になる
  - XDDPからの学びの例：理由を書く、Before/Afterを書く

- コーディングを最後まで遅らせる：違いを感じる点
  - 本当に動くかどうかの確認が遅れる。
  - 仕様の理解、設計の質などへの学びのフィードバックが遅れる。
  - 変更設計書も仮説であり、想定どうりになるかどうかはわからない
  - 顧客へのリリースとそのフィードバックも遅れることで、作っているものの価値の評価も遅れる

まとめてみると

派生開発推進協議会
AFFORDD

- アジャイル開発は、派生開発といえる
  - 扱っている問題は同じ
  - シチュエーションも同じ
  - プロセスの面から
    - 問題領域は両者抑えている
      - 要求を抑えようとしている
    - 解決領域は違う
      - XDDP
        - ドキュメントとプロセスの工夫
      - アジャイル
        - 小さく、コードをテストしながら、積み上げることを繰り返す

ディスカッション3

さて、どうしましょうか

あなたは、コードを遅らすことと、
コードを要求ごとに動くかどうか評価すること

どちらがよいとおもいますか？

システム更新の二つの方法

派生開発推進協議会
AFFORDD

準備をして変更する

業界標準

実行しようとする変更作業を注意深く計画し、
更新対象のコードを理解し得散ることを確かめ
たうえで変更に着手します。
注意深く作業し、とてもプロらしい方法

保護して変更する Cover and Modify

保護＝テスト
変更に問題があった場合でも、ソフトウェアの
ほかに害が及ばないようにする

テスト

派生開発推進協議会
AFFORDD

レグレッションテスト

既存コードを作るときに作ったテストを行い、
影響が起きてないことを確かめるテスト

シフトレフト

バックログ（要求）を実装したら、
すぐにテストをする

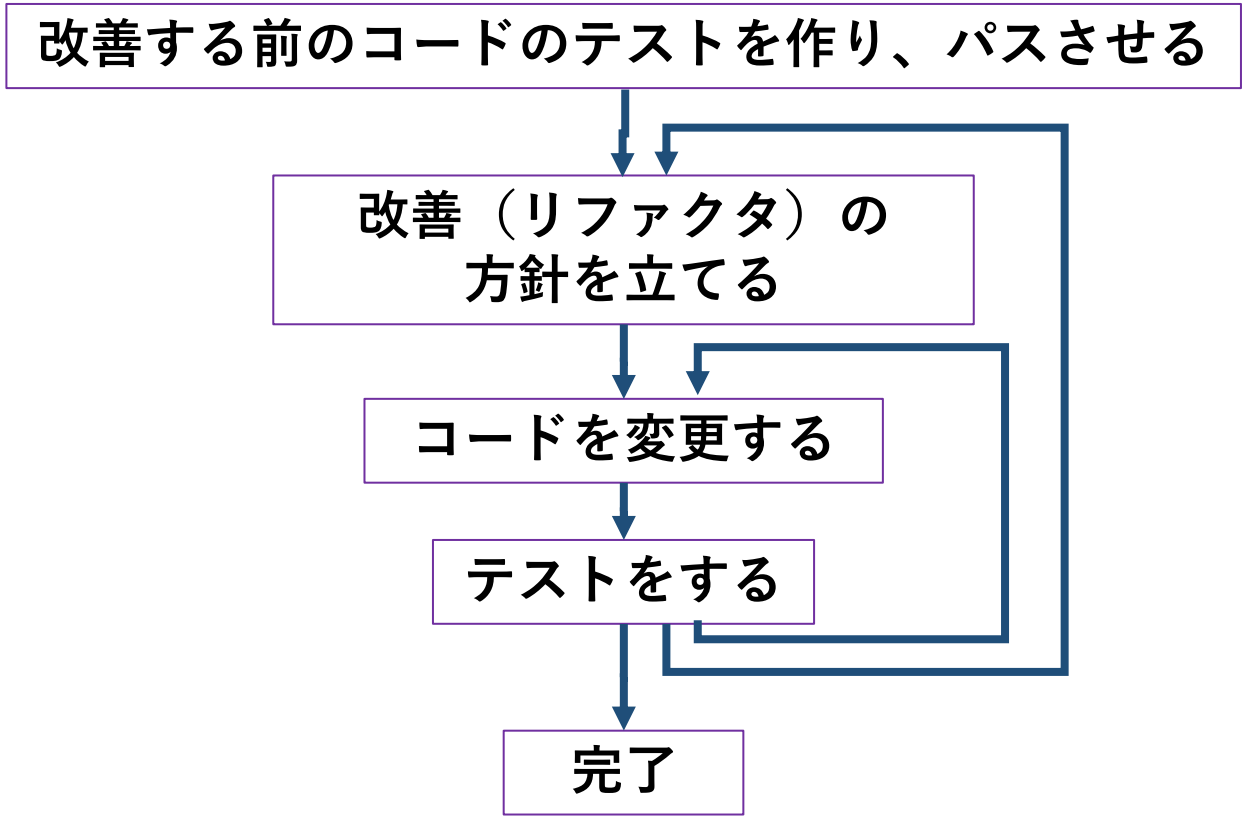
TDD

設計時にテストの網を張りながら
開発をする

派生開発推進協議会
AFFORDD

マーチン・ファウラー

コードを改善するプロセス

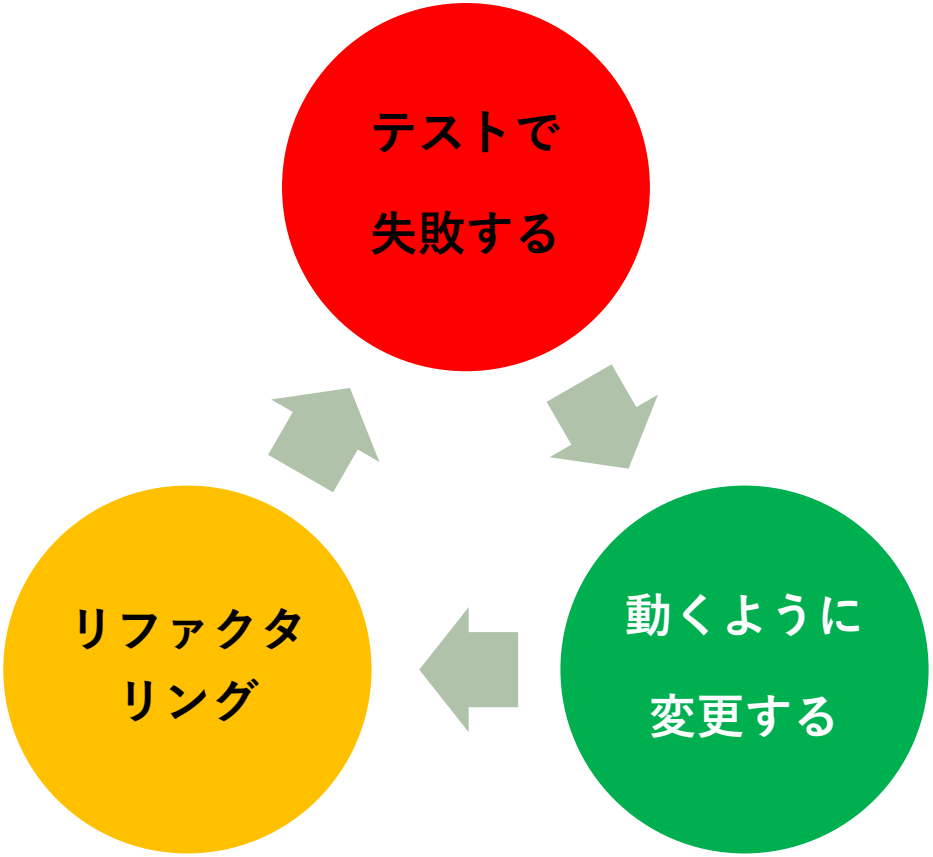


TDD テスト駆動開発 おすすめの工夫

派生開発推進協議会
AFFORDD

ケント・ベック：ソフトウェアを構築する方法

機能させる：Make it work
正しくする：Make it right
速くする：Make it fast



アジャイル開発の進化

多種多様のアイデア

多くの手法、工夫、プロセス、考え方
組織の在り方、チームのつくり方

そして、アジャイルを進化させてきた

コミュニティが支えた

派生開発推進協議会
AFFORDD

派生開発を
進化させよう

Afforddは進化の場

派生開発推進協議会
AFFORDD

ディスカッション4

では、どんな進化が考えられるでしょうか

皆さんの派生開発の進化した姿を妄想してください

派生開発推進協議会
AFFORDD

おしまい

ありがとうございました
T6一同

T4（テスト）T6(アジャイル）の研究会に参加してみませんか？