無秩序が派生開発の混乱を招く

―ＸDDPで品質と生産性を同時に確保 ―

第４回組込みシステム技術セミナー講演資料

（社）組込みシステム技術協会　中部支部

2013.3.1

派生開発推進協議会
代表　清 水　吉 男
URL=http://www.xddp.jp
株式会社 システムクリエイツ　代表取締役
URL=http://homepage3.nifty.com/koha_hp
shimz@nifty.com

自己紹介

派生開発推進協議会：AFFORDD

- 2010年２月に「派生開発推進協議会(AFFORDD)」(http://www.xddp.jp/)を設立し、「XDDP」「USDM」「PFD」などの派生開発に有効な手法の普及活動を開始

カンファレンス

- 年１回開催、一般公募
- 現場での成果／研究会の成果を発表

研究会／フォーラム

- 現在１０の研究会が稼働
- XDDPやUSDMに対して、入門から効果的な活用方法について研究活動
- 研究会の成果をカンファレンスとは別に知らせる
- 次の指導者の育成

勉強会

- XDDP，USDM，PFDを多くの人に知ってもらうために、首都圏以外に地方でもセミナーや勉強会を繰り返し開催する

地方部会

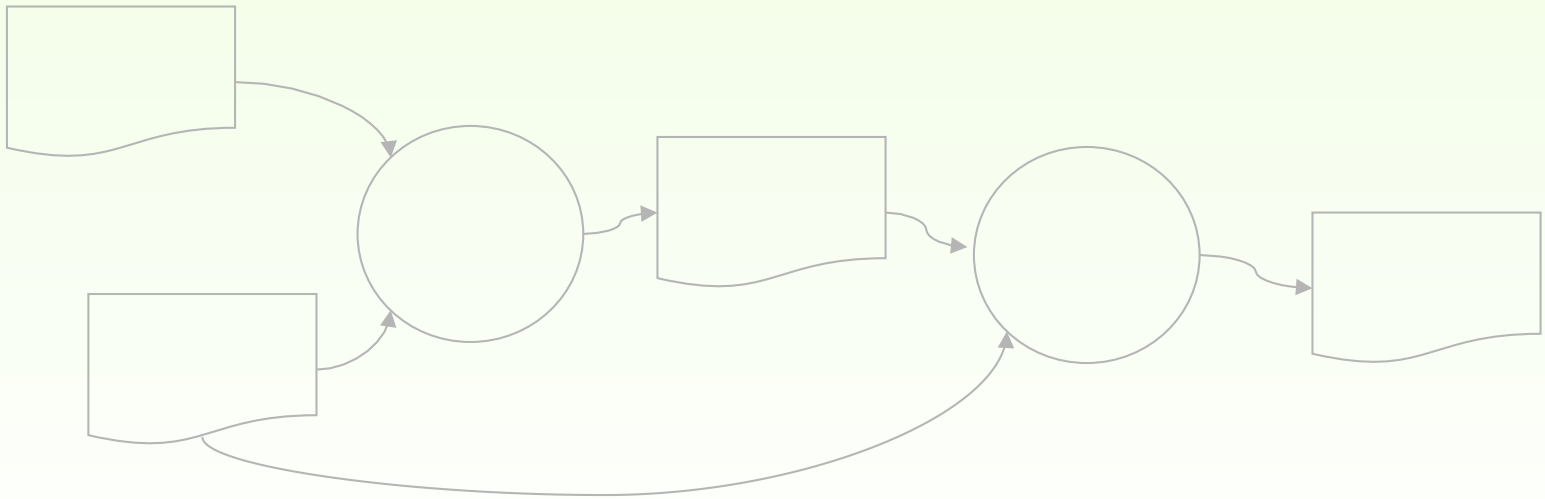
- 各地方単位で会員がまとまって情報交換や技術交流をすすめる

地方支援

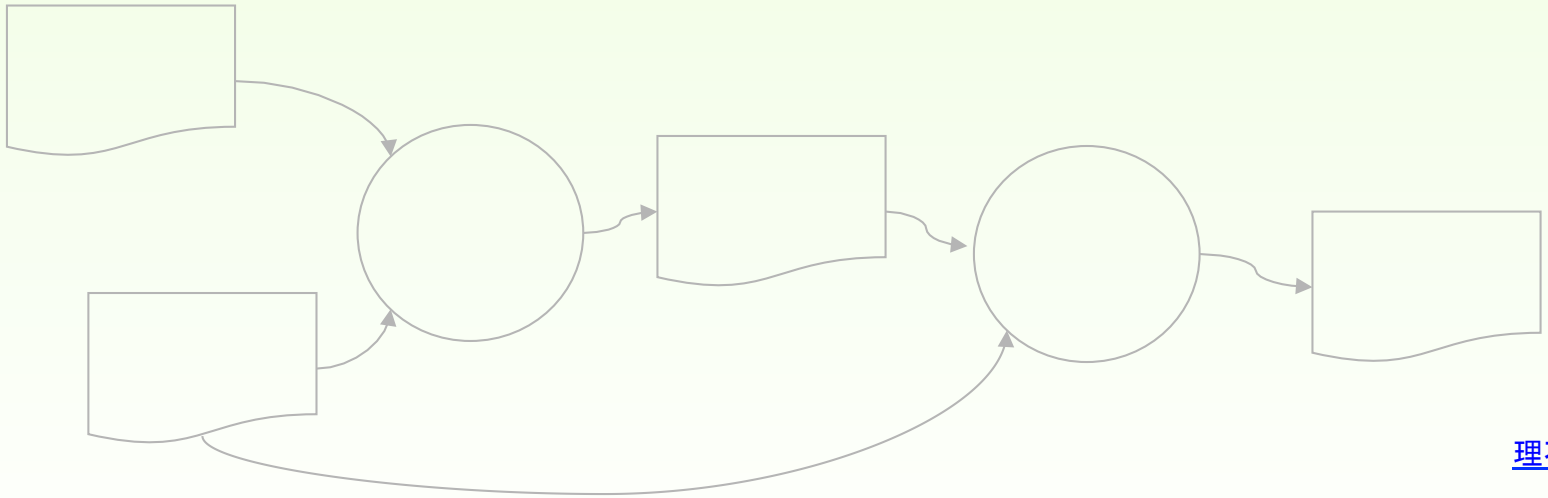
- 地方を強くするという考えのもとに、行政の活動を支援する

agenda

- 派生開発の現場
- 派生開発の特徴
- XDDPの要点
- XDDPとアジャイル



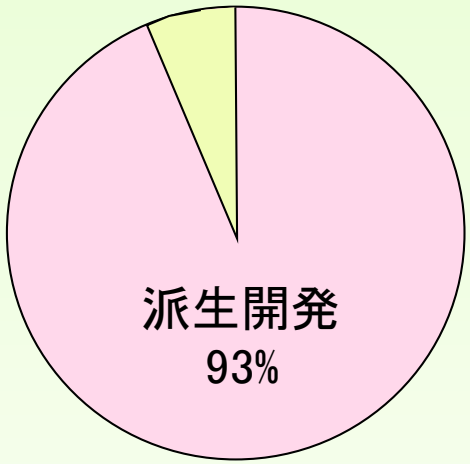
1. 派生開発の現場
2. 派生開発の特徴
3. XDDPの要点
4. XDDPとアジャイル



理不尽な世界

大部分は派生開発

- 組織によっては、開発案件の90%以上は「派生開発」



ある企業の1部門のケース

- 入社後、一度も新規開発に関わらない可能性大
- 一般に公表されている開発方法は「新規開発向け」
- 組織の標準プロセスも派生開発にマッチしていない

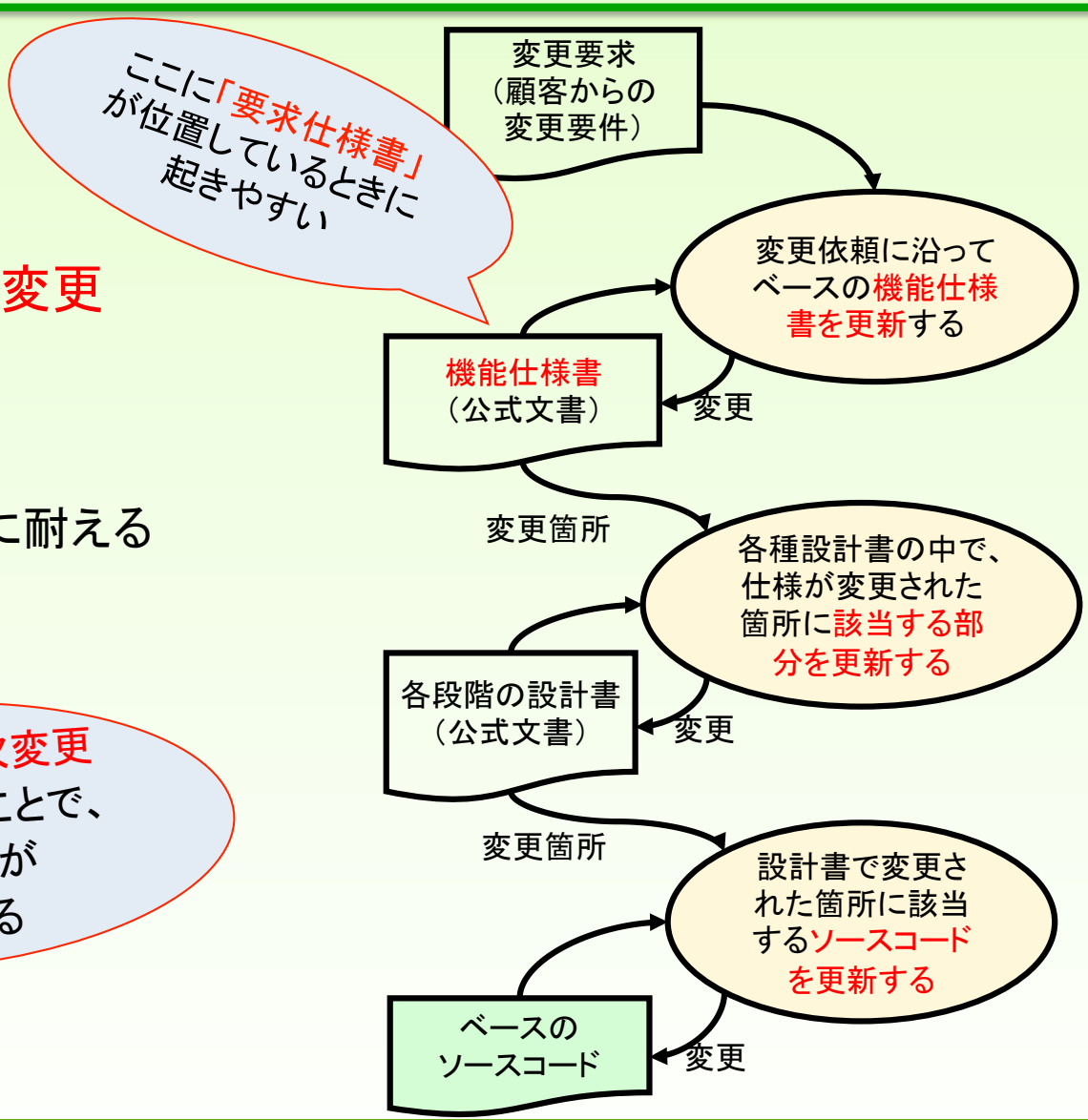
いろいろな問題が発生する

公式文書があるケース

“新規開発崩し”の変更

- 「新規開発崩し」
  - 機能仕様書から順に
    文書や ソースコードを変更
    （完成）していく.
  - 公式文書も、必ずしも変更に耐える
    ようには書かれていない

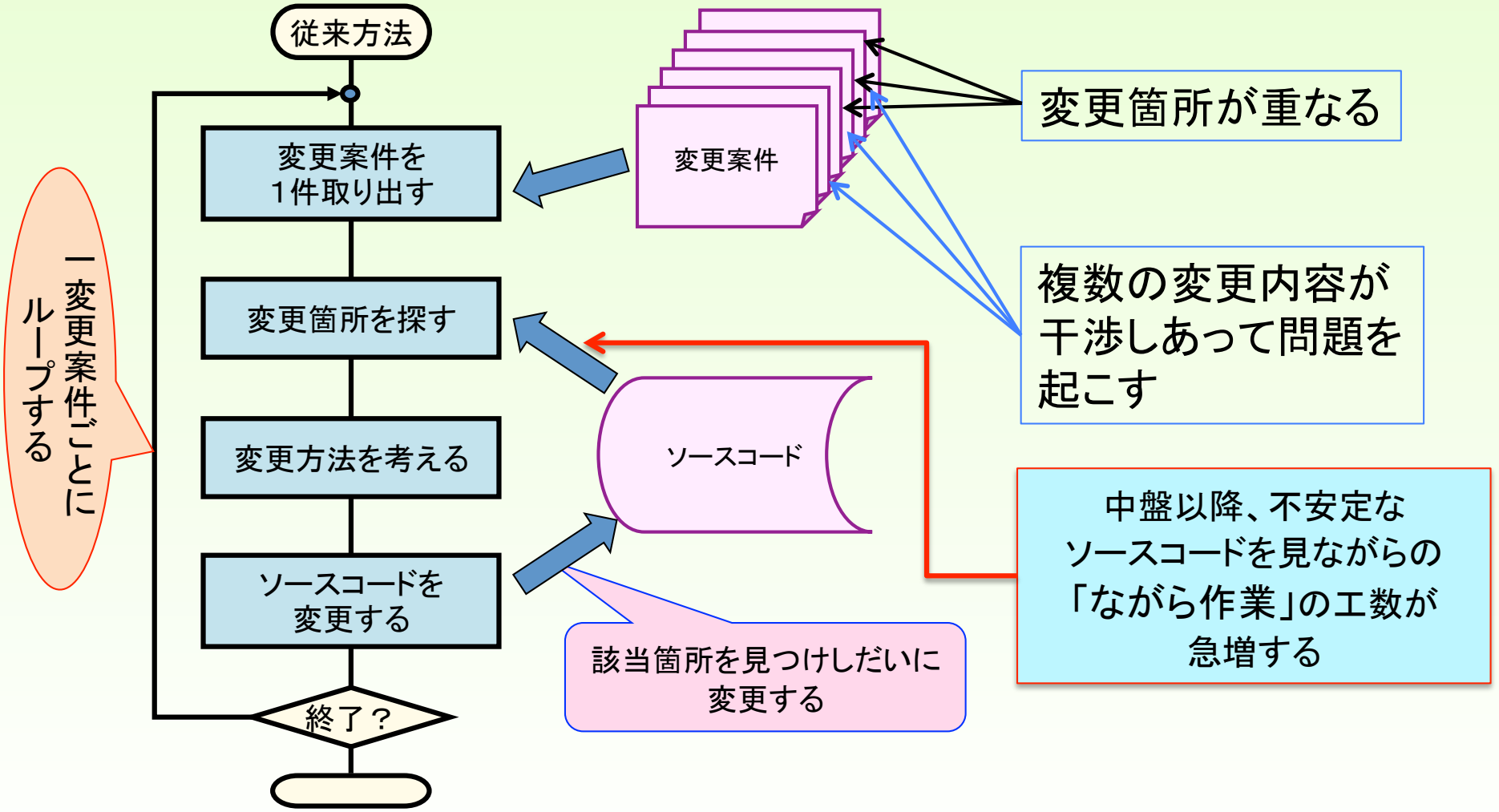
公式文書に逐次変更
が織り込まれることで、
他への影響が
見えなくなる



公式文書がないケース

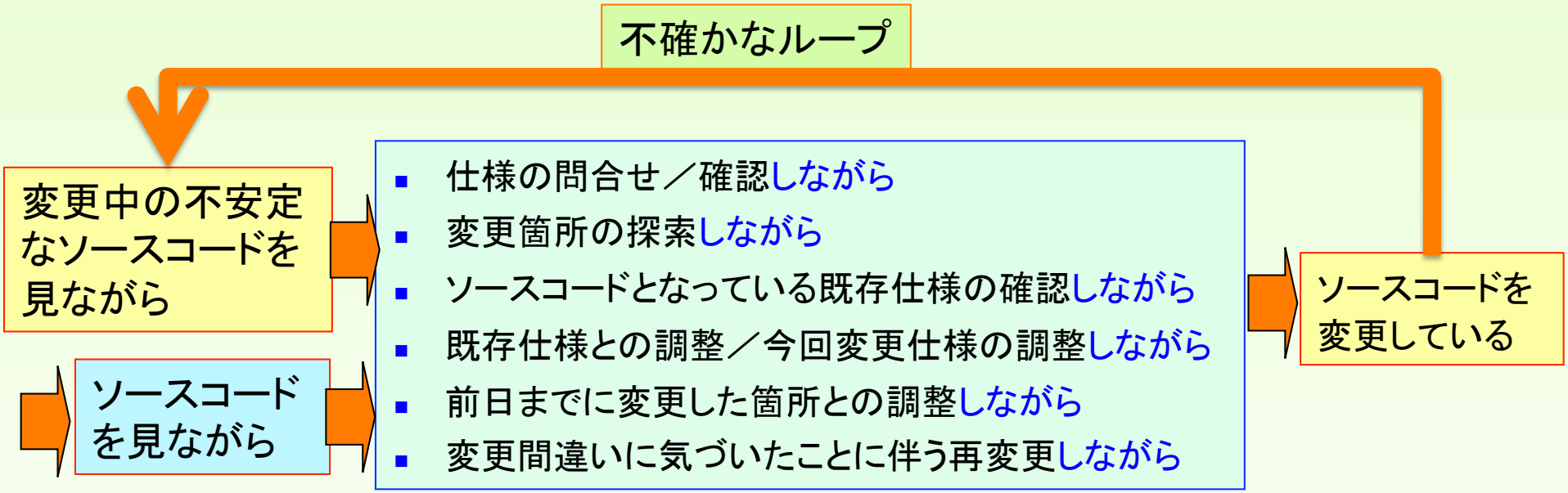
見つけ次第に「いきなりの変更」

- 公式文書が残されていない多くの現場でのパターン



「ながら作業」が増える

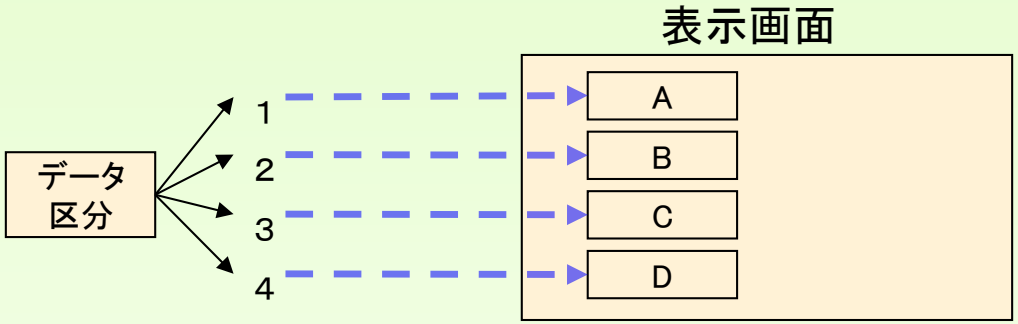
- 変更情報を書き留める手段がない　＞「ながら作業」が増える



中盤以降からこのような「ながら作業」の工数が一気に増加する

- その結果・・・「実装プロセス」の生産性も　10行未満／時間　となる

変更の実現方法も1つとは限らない（1）



- 変更前
  - データ区分（1～4）に応じて金額の表示欄が「A～D」に対応していた

変更依頼：
　区分が2で「C1」の条件のときは表示の位置をB欄からC欄に変更してほしい

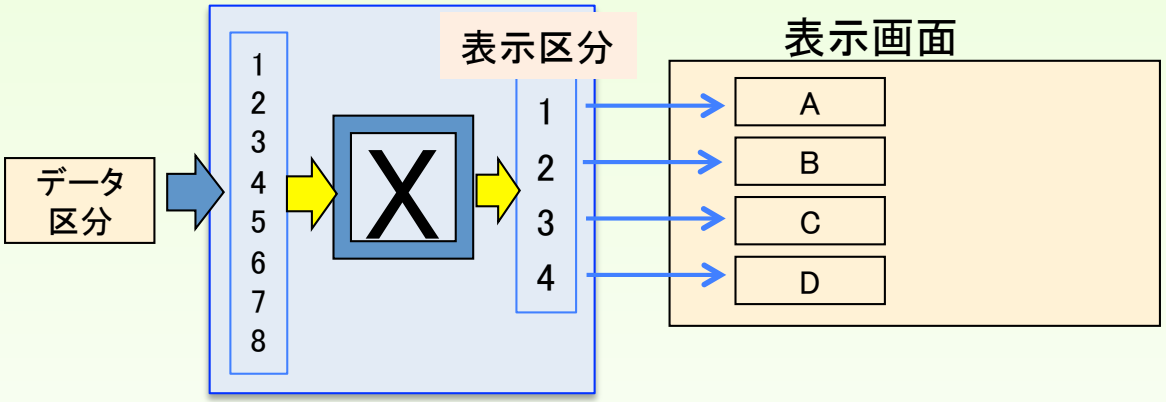
表示処理のところで条件を判断して表示の場所を「B」から「C」に変更する

この変更は悪くないが・・・

変更の実現方法も1つとは限らない (2)

2年後:
　表示欄は4種類のままで区分が8種類に増えた

「表示区分」を導入して、データ区分を表示区分に変換して対応

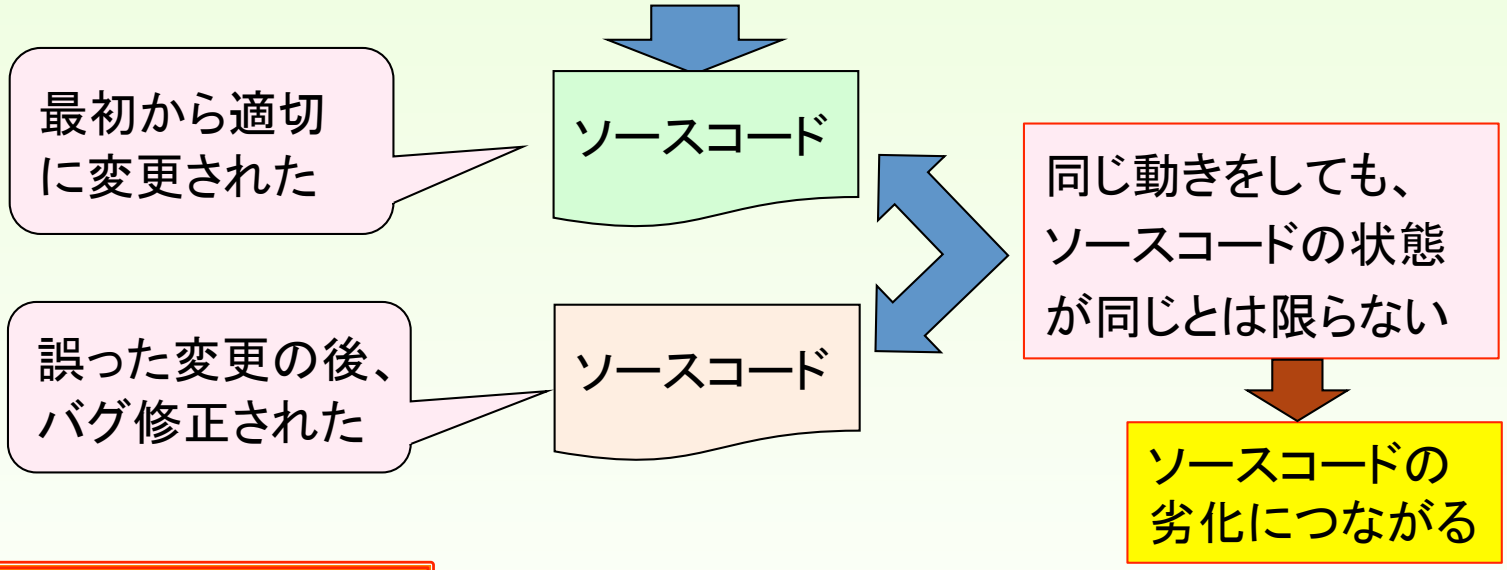


2年前に表示処理の中で対応した変更がバグとなって表面化

この問題を未然に防止する方法は？

バグが見つかったら修正すればいいのか？

- 安易な考え方でソースコードを変更しているので、次々とバグが発生する
- バグを発生させたのと同じプロセスでバグを修正するため、バグの連鎖が止まらない



不確実にソースコードを変更することの弊害

あとでより良い変更箇所が見つかったときは？

見つけ次第にソースコードを変更していくなかで、
後になって、より良い変更方法に気付くことがある

- この時、変更方法を変更することができますか？

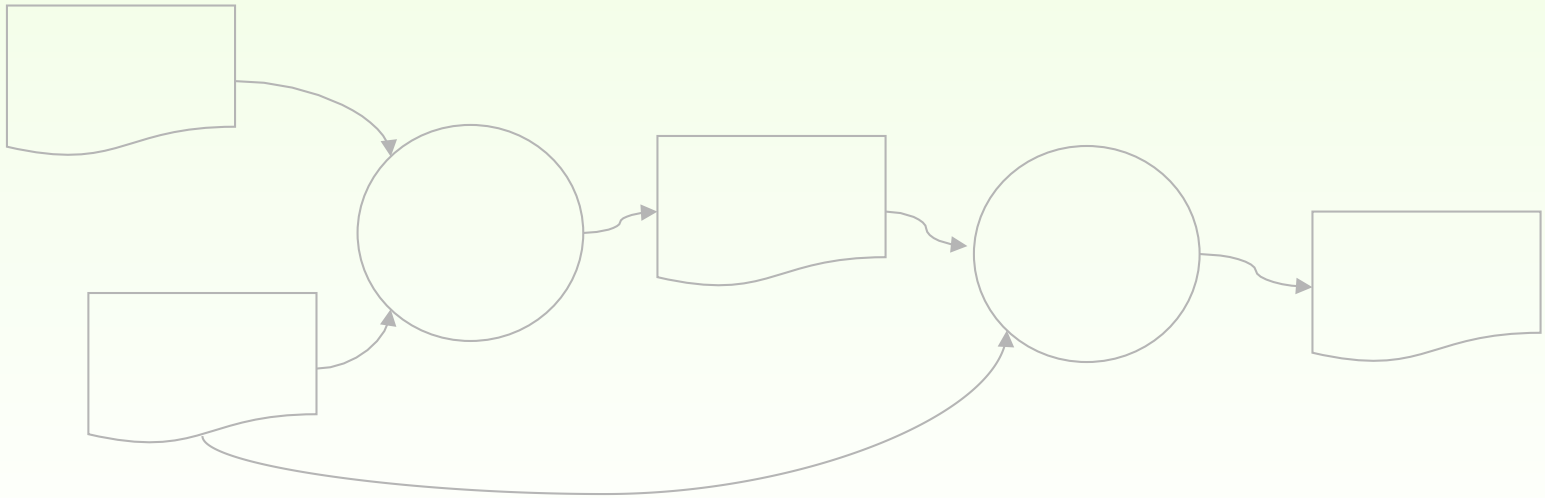
先の変更方法でも
間違いではない

→ 変 更 方 法 を 変 更 し な い

「気付かなかったことにする」ことの問題の重要性
（人格の毀損）に気付いて欲しい

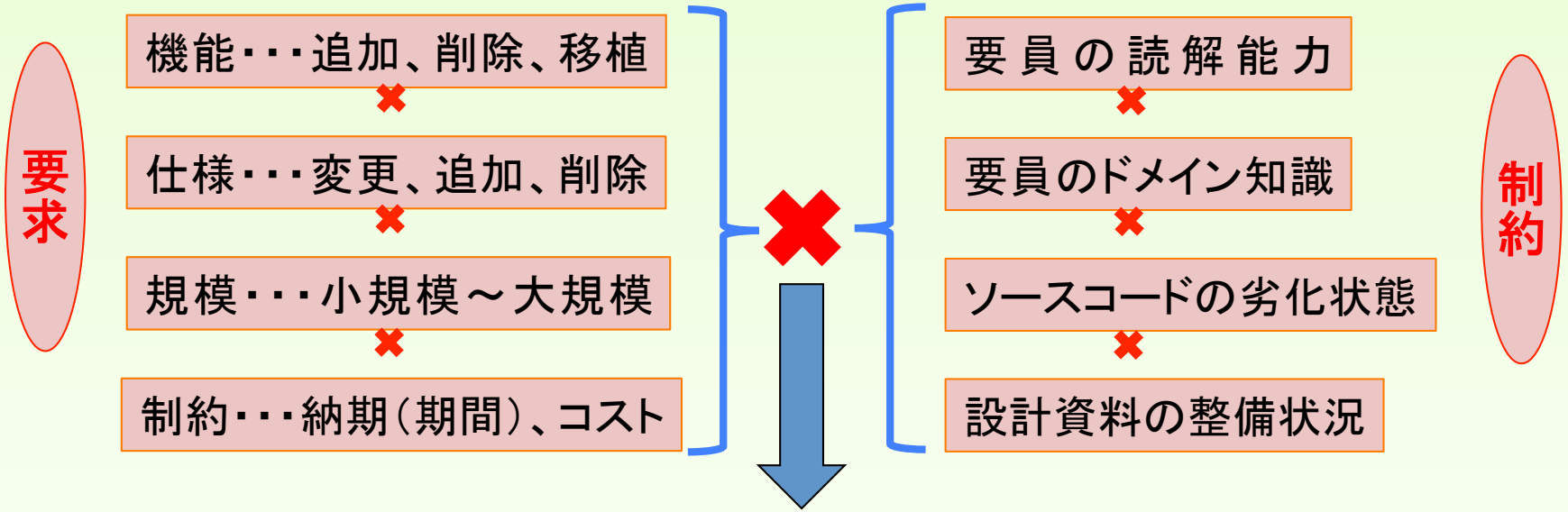
不確実にソースコードを
変 更 す る こ と の 弊 害

1. 派生開発の現場
2. 派生開発の特徴
3. XDDPの要点
4. XDDPとアジャイル



派生開発の要求は多様

- 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



わずか1000行の変更だけの１人プロジェクトと、10万行の機能追加と変更が混じった10人のプロジェクトが、同じプロセスで実現できますか？

派生開発は「部分理解」の制約を受ける

- 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れない

- 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

↓

- 担当者の「思い込み」「勘違い」が混入する

新規開発と派生開発のバグの違いの意味

- 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

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

「変更」のバグは、新規開発や追加のプロセスでは対応できない

レビューの視点も変えるべき

派生開発の主要な特徴

- 変更前は（問題なく）動いていた
  - 今回、変更を加えたことで動かなくなった
  - 原因は「今回の変更」にある）
- デグレードの問題が発生する
  - 変更仕様同士で変更内容が干渉し合う
  - 変更仕様に関連した箇所の変更もれ（関係箇所）
  - 変更内容に対して思わぬところで影響を受ける（影響箇所）

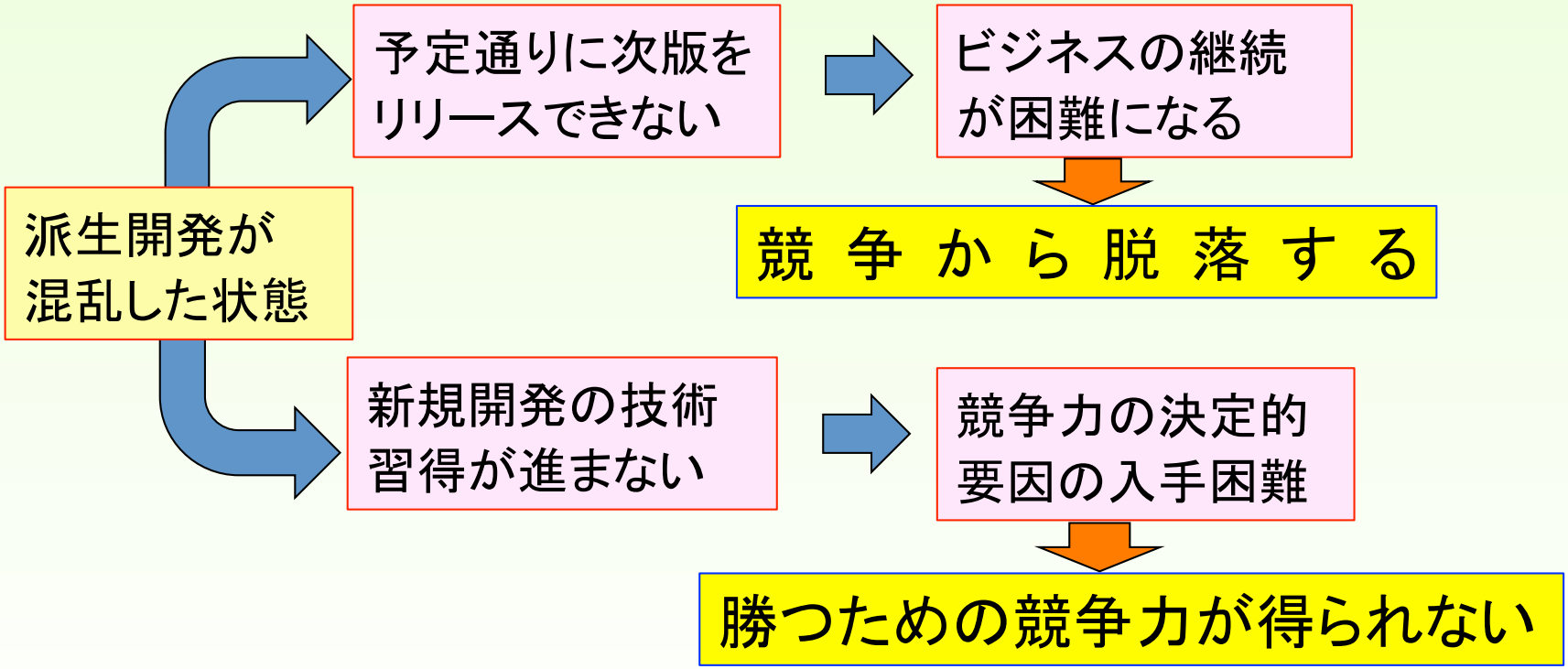
これをどうやって
未然防止するか

このままではリリースできない事態も

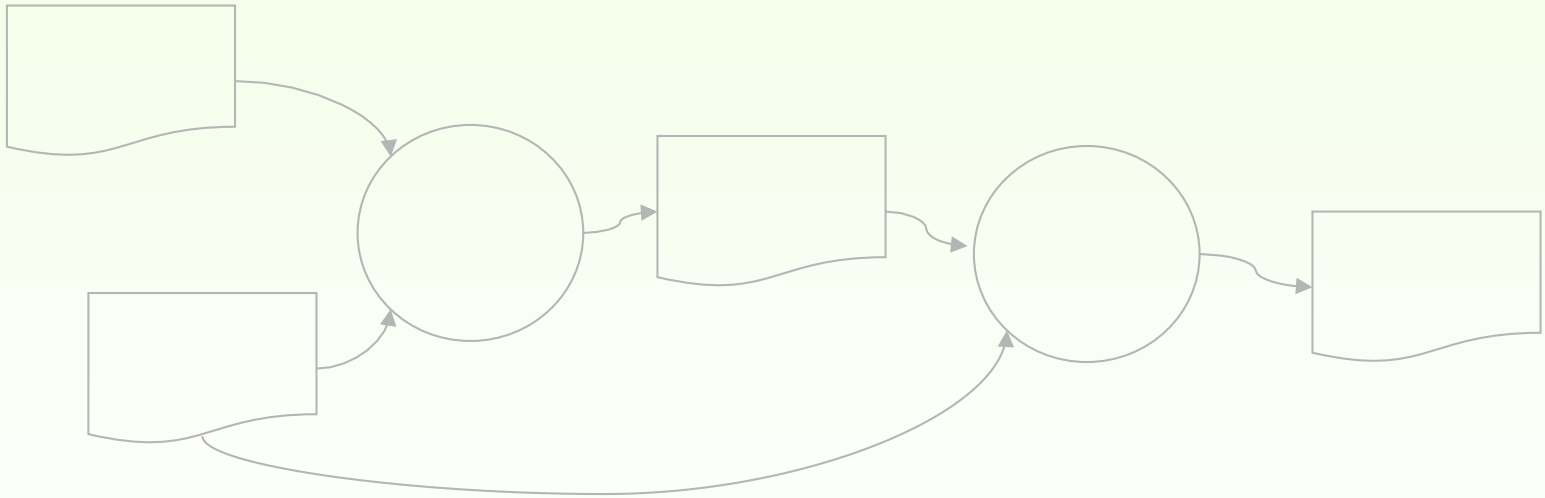
派生開発で現行の「ソースコード」を回し続けるしかない？

現行の「ソースコード」はいつまで競争力を維持できるのか？

- 派生開発にうまく対応できなければ事業に行き詰まる危険も



1. 派生開発の現場
2. 派生開発の特徴
3. XDDPの要点
4. XDDPとアジャイル



「XDDP」の誕生

４０数項目の機能追加と変更を３ヶ月で！

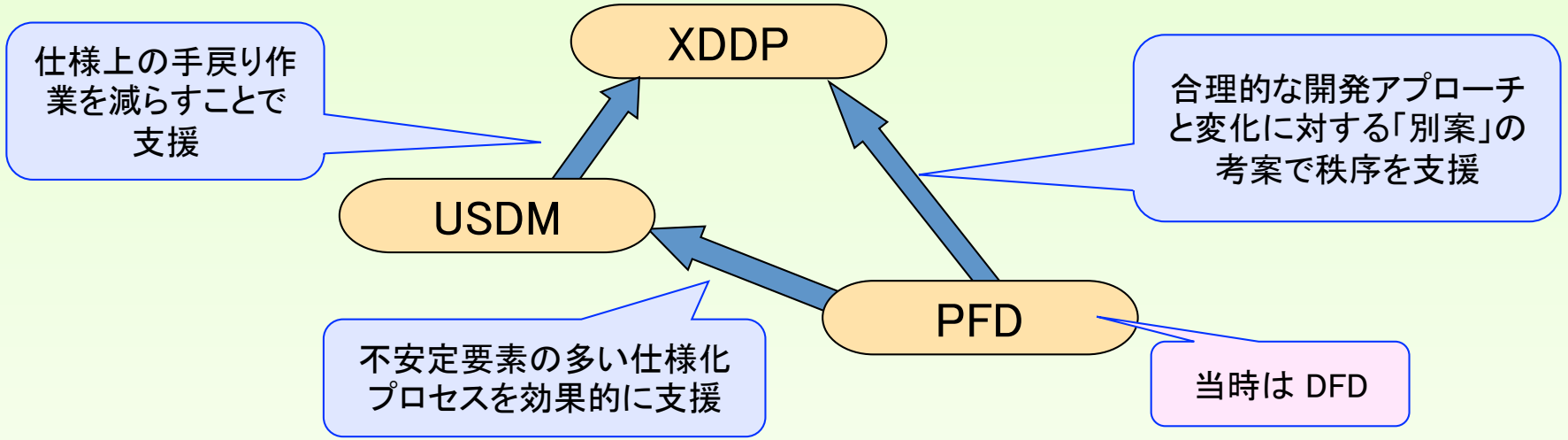
- アメリカの顧客からの要求（1978年）
  - 初めてのドメイン、初めて見るソースコード、言語も？
  - 国内の客先には、その製品の仕様を知る人はいない
- 「保守のプロセスでは不可能」と判断
- １週間で、新しいプロセスを設計してシミュレーションで確認
  - 機能追加と変更の２種類のプロセスに分けて品質と生産性を確保

品質	変更箇所（理解した仕様を含む）を「before/after」で記述し、Faxでレビューを依頼
	「before」は、現状の仕様を私が理解した状態 → レビューで確認
生産性	必要不可欠な最小限の成果物とプロセスの連鎖で構成

３ヶ月の納期を達成

XDDPトライアングル

- 「XDDP」は、「USDM」と「PFD」の支援の上で成り立っている



PFD	ムダのない合理的な開発アプローチを設計するための技術
USDM	要求仕様を書くための技術 「追加機能要求仕様書」「変更要求仕様書」のベースとなる
XDDP	派生開発のプロセスを組み立てる技術 「変更３点セット」を中心に派生開発をスムースに推進する

USDMにおける「要求仕様書」の考え方

USDMにおける定義

「USDM」では、要求仕様書とは、今回のプロジェクトで実現して欲しいこと（Requirements）について、“作ることの関係者”が実現内容についての認識を特定（Specify）できている文書

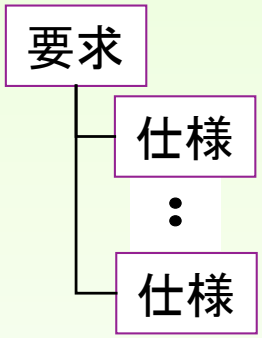
「USDM」では「Specification」とは関係者が同じものをイメージできる状態のことと捉える

	要求仕様書	機能仕様書（等）
目的	作るためのもの	機能を説明するもの
関係者	計画書で特定されている	特定されていない
表現	関係者がSpecifyできる状態	一般的な記述
納期やコスト	背後に背負っている	意識されない
バージョン管理	今回だけの文書	ソースコードと対でバージョンアップする

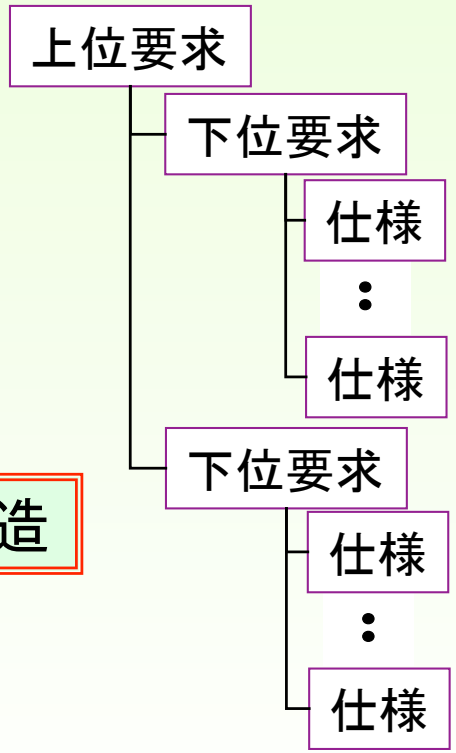
要求と仕様を階層構造で捉える

- USDMでは「要求」と「仕様」を階層構造の中で捉える
  - 上位要求の範囲が広いときは、要求を階層化する

① 要求と仕様の階層



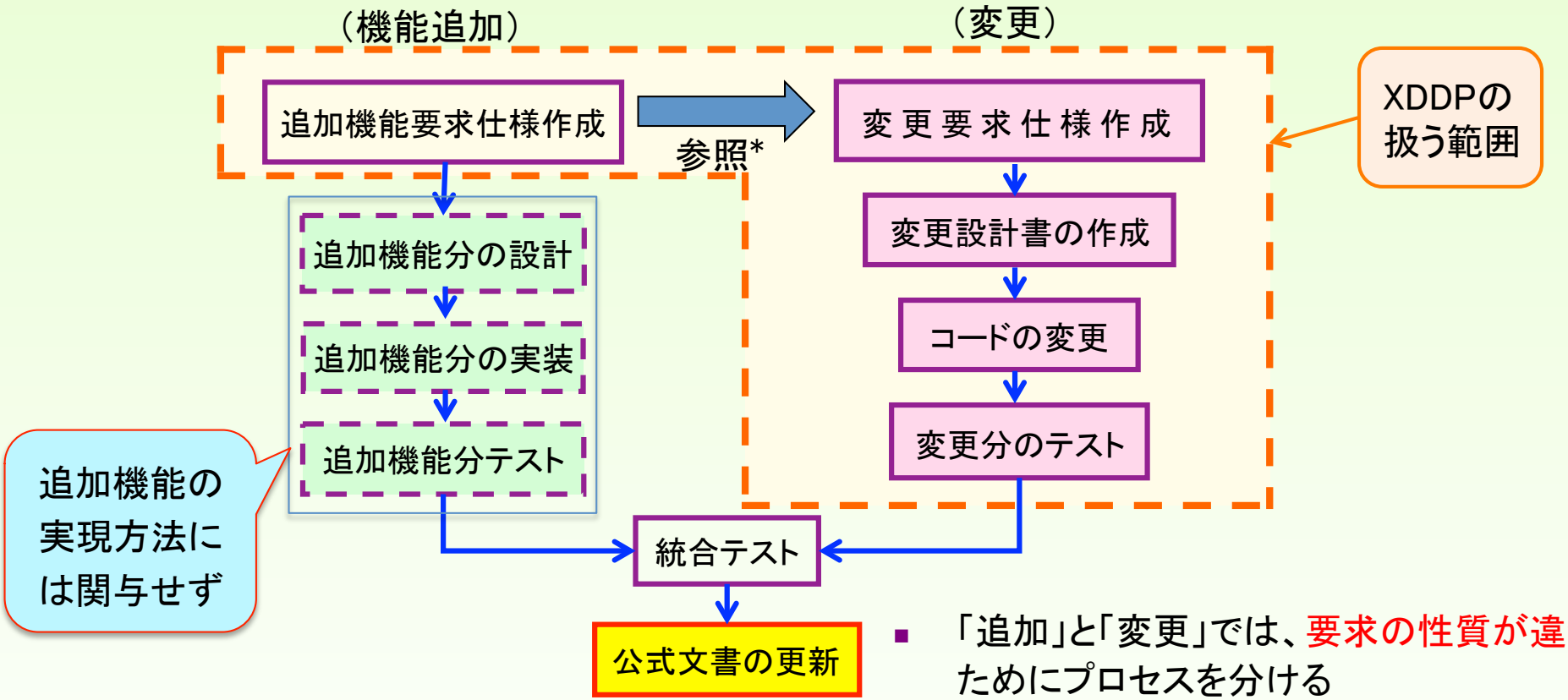
② 上位要求と下位要求の階層



階層構造＝仕様がモレにくい構造

XDDPは2種類のプロセスを並行させる

- 「XDDP」では機能追加と変更を異なるプロセスで対応する



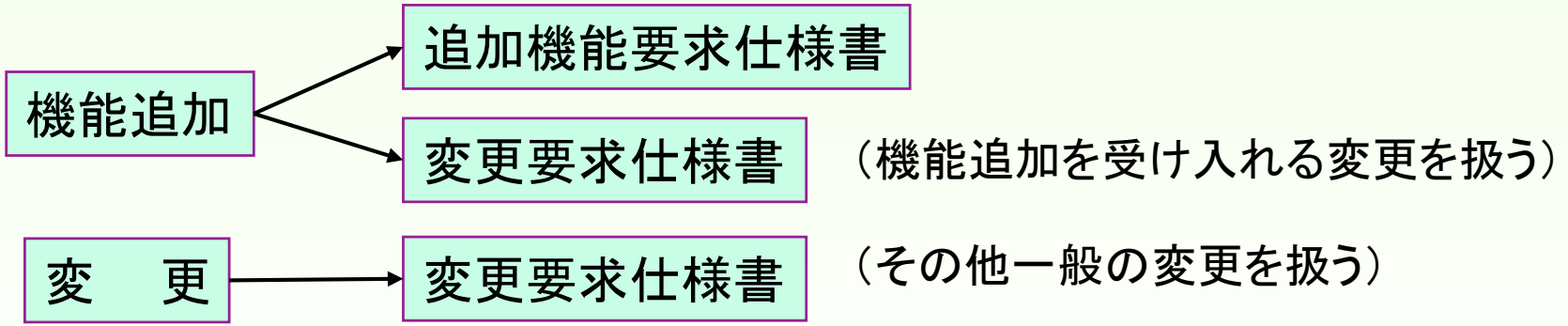
＊追加機能を受け入れるための
変更方法を探すために参照する

- 「追加」と「変更」では、要求の性質が違うためにプロセスを分ける
- 「変更」は「差分」で進める
- 公式文書の更新はテスト後（後半）に行う

派生開発では2種類の要求仕様書が必要

- 要求仕様書が異なる以上、対応するプロセスも変わる

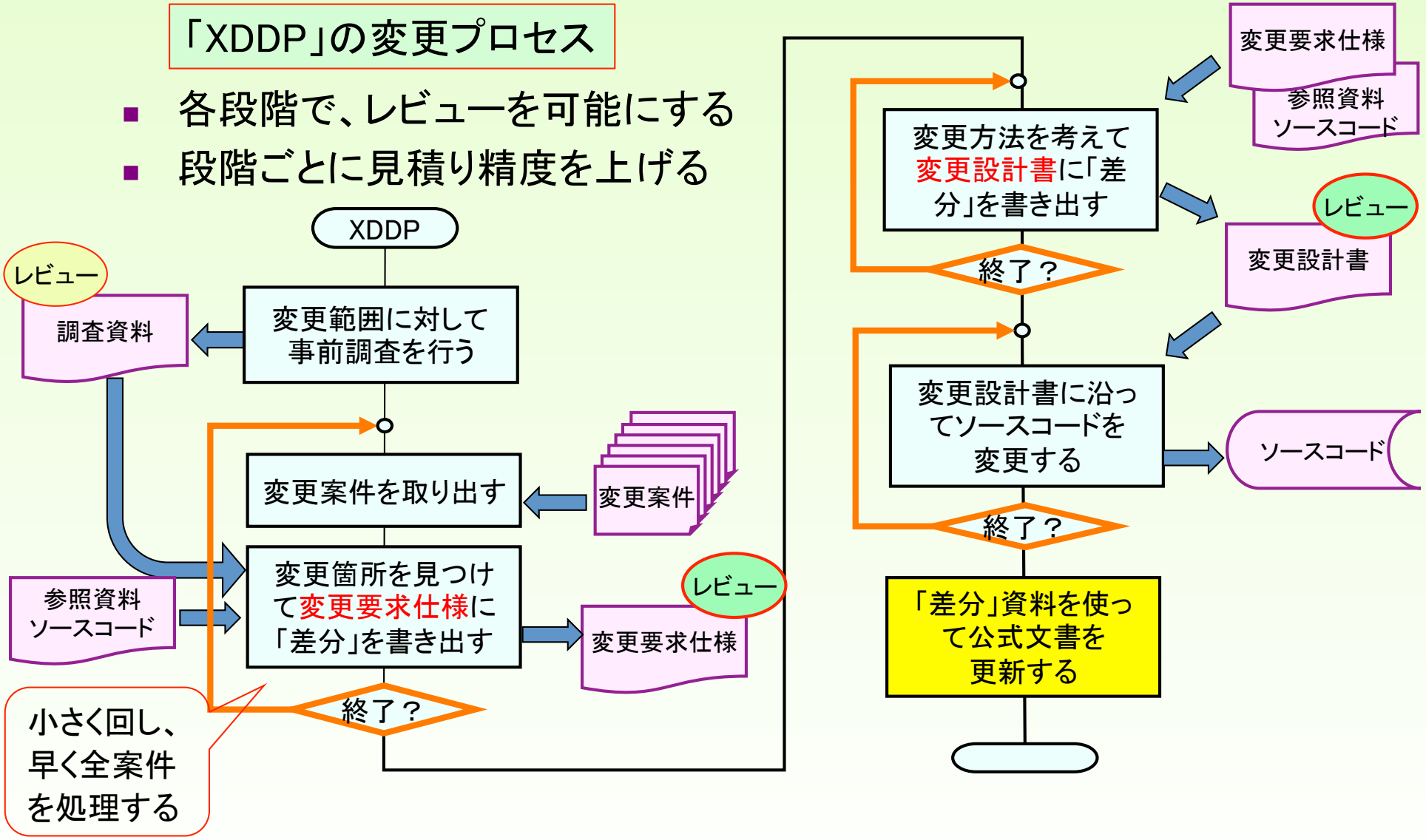
	要求	対　応	要求仕様書	対応プロセス
機能レベル	追加	機能追加として扱う	追加分＝追加機能要求仕様書	追加用プロセス
			変更分＝変更要求仕様書	変更用プロセス
	移植	通常は変更として扱う	変更要求仕様書	
	削除	変更として扱う		
仕様レベル	追加	変更として扱う		
	変更			
	削除			



変更は成果物に対して小さく回す

「XDDP」の変更プロセス

- 各段階で、レビューを可能にする
- 段階ごとに見積り精度を上げる



「追加機能要求仕様書」

- 派生開発における追加機能を扱う要求仕様書

USDMにおける定義

今回のプロジェクトで追加実現したいこと（Requirements）について、特定された関係者がその機能の内容を特定した（Specify）ことがまとめられた文書

「Specification」とは関係者が同じ物をイメージできる状態のことと定義する

- 追加機能の仕様（実現して欲しいこと）が記述されている文書
- 実現方法は「設計プロセス」で選択され評価される
- 作るための文書だから「作り方の品質要求」も含まれる

「USDM」の基本的考え

- 機能要求は一般に「振る舞い」で記述できる

機能の「仕様」は要求に含まれる「動詞」（および目的語）に存在する

要求は「理由」とセットで表現する

- 要求には、それが必要な「理由」（存在理由）や「背景」がある
  - その「要求」がないと困ることや恩恵を受けることは？
  - 「理由」によって要求に対する認識のズレを調整できる

◆ 「理由」は「要求」の一部

要求	SAL01	商品毎に設定されている当日の売り上げ数量の予測に対して、実売データとの間に大きな開きがあるときは警告メッセージをマネージャーのPC画面に出す
	理由	すぐに原因を調べて、陳列方法の変更など適切な対応策を講じる必要がある

「要求」を表現し、「理由」を捉えることで、
顧客が本当に望んでいることを実現する

- 単に、機能を実現するだけでは「価値」はない！
- どのように、どのタイミングで表示すればこの機能の「価値」が上がるのか？

分割・階層化して要求の範囲を狭める

- 要求の範囲が広い（動詞が多い）ときは、上位要求を分割階層化する

要求	MAL01	事前に指定された受信および送信した電子メールをキーワードで検索して、選択した電子メールをメーラーに繋いで再利用したい
	理由	メールが多くて、関連するメールを 探すのに手間取る

	要求	MAL01-01	表示された検索グループの中から一つを指定する
		理由	検索グループが複数あるから
	要求	MAL01-02	いくつかのキーワードの入力を受付毛、それらを組み合わせて検索する
		理由	可能性のあるキーワード（複数）で探したい
	要求	MAL01-03	検索結果を表示し、見つかったときは「subject」などの情報を一覧で表示する
		理由	Subjectと日付などから目的のメールを探すことになる
	要求	MAL01-04	一覧の中から選択されたメールを開く
		理由	中身を見ないと分からないから
	要求	MAL01-05	一つのメールを開いた状態でメーラーに繋いで編集できるようにする
		理由	コピーの手間を省いて編集の操作に入りたい

事前に設定されたグループの仕様を埋める

- 範囲が狭められているので、容易に抽出できる

	要求	MAL01-03	検索結果とメールのSubjectを表示し、選択された内容を表示する
		理由	目的のメールが一つとは限らないので、絞り込めるような操作がしたい
	□ □ □	＜検索結果の表示＞	
	□ □ □	MAL01-03-1	検索されたメールの件数を一覧の上に表示する
	□ □ □	MAL01-03-2	該当するメールが存在しないときは「該当無し」を表示する
		＜検索メールの表示＞	
	□ □ □	MAL01-03-5	検索されたメールの「Subject」を一覧で見せる
	□ □ □	MAL01-03-6	検索されたメールに連続番号をつけて表示する
	□ □ □	MAL01-03-7	検索されたメールの件数が10件を超えるときはスクロールバーを見せる
		＜メールの中身の表示＞	
	□ □ □	MAL01-03-10	一覧から1つのメールを選んで、その内容を見ることができる
	□ □ □	MAL01-03-11	開示されたメールの中で検索キーワードと一致している文字列を赤色で表示する

仕様化作業に「生産性データ」が使える

- USDMの表記法は、＜仕様グループ＞に対する仕様化作業に「生産性データ」が使える
- 仕様化作業は、実際に設計する人を投入できる
  - 「50仕様／H」のスキルがあれば
  - 5000仕様・・・100時間　＞　5人投入　＞　20時間で書ける！！

仕様化作業の工数が見通せている ➡ さらに「価値」を盛り込む

- 生産性データの効能
  - 仕様化作業の日程を約束できる
  - 要求の表現や ＜仕様グループ＞が適切に表現できていないときは、想定している生産性を発揮できないので、直ぐに改善する

仕様変更率“5%以下”を目指す

- データを使って仕様の表現が適切かどうかを知る

① 要求仕様のレビューの指摘件数(指摘率)
② 仕様関係のバグの発生率(KLOC単位)
③ 要求仕様のベースライン設定後の仕様変更率
④ 仕様の問合せ件数(率) など

ベースライン設定後の仕様変更率

$$\left(\frac{\text{変更仕様数}}{\text{総仕様数}} \times 100\right) \leqq 5\%$$

- 5%以下になることの効果
  - 要件管理での対応が容易になる
  - 追加機能の受け入れのための変更が混乱しない
  - 次回以降にこの機能の仕様変更が混乱しない

「変更要求仕様書」

- 「XDDP」では、すべての変更を扱う仕様書として「変更要求仕様書」を導入する

「要求仕様書」の概念から発展したもの

今回の派生開発で変更したいこと（Change Requirements）について、特定された関係者が変更内容まで含めてその内容を特定（Specify）したことがまとめられた文書

- 「変更要求」と「変更仕様」を階層関係で捉える

- 変更プロセスには、いわゆる「設計プロセス」が存在しないため、ここで変更の実現の様子を見せる
- 「変更仕様」として関数仕様を変更するようなレベルで記述することで、関係者が「Specify」できるようにする

「変更３点セット」の成果物の意味

- 「XDDP」の変更プロセスでは3つの成果物を作成する（必須）

成果物	カバー範囲	記述内容	レビュー機会	
変更要求仕様書	What （Why）	何を変更するか？ どのような振る舞いを変更するか なぜ、変更するのか？	○	○
TM（Traceability Matrix）	Where	変更する仕様がどこにあるか？	○	
変更設計書	How	具体的な変更方法を記述する	○	○

- 効果的なレビューの機会を確保して思い込みや勘違いをカバーする
- ソースコードの変更作業と、公式文書のマージの両方に活用できる
- 今回の派生開発の変更記録として保存する

「before」「after」で表現する

- 変更要求、変更仕様は、必ず「before」「after」で表現する
  - 「before」は現状を表現し、影響箇所に気付かせる効果が大きい
    - ソースコード、各種仕様書・設計資料の該当カ所を探すのに有効
  - 「追加する」「削除する」は、それ自身に「before」「after」を含んでいる

この記述から「関係箇所」「影響箇所」に気付く

変更要求	DSP.11	全ての画面表示の日付の表示を西暦の年号に統一して欲しい
変更要求	TIM.05	予約機能の時間表示を円形表示から棒線表示に変更したい
変更要求	MEA.01	計測状況をリアルタイムで表示する機能を追加したい
変更要求	SEC.08	ビル入館システムにある個人認証機能を、このシステムに移植して欲しい

変更要求は「範囲」と「理由」を表現する

- “変更要求”も変更が及ぶ「範囲」を表現することで、「要求」としての役割を果たす
  - 適切に「範囲」が伝わらなければ、変更モレが発生しやすくなる
- 変更する「理由」を付けて範囲の特定を支援する

要求	LST.01	商品一覧と在庫確認リストの商品名の横に写真の表示を追加する
	理由	派生品が多くなったことで、商品名を間違えるケースが増えているから

要求	DSP.01	受信した計測データの横のアルチ画面表示あら、縦に並べて表示するように変更する
	理由	各形測地点での相違を時系列で一目で見えるようにして、同期のずれを早期に発見したい

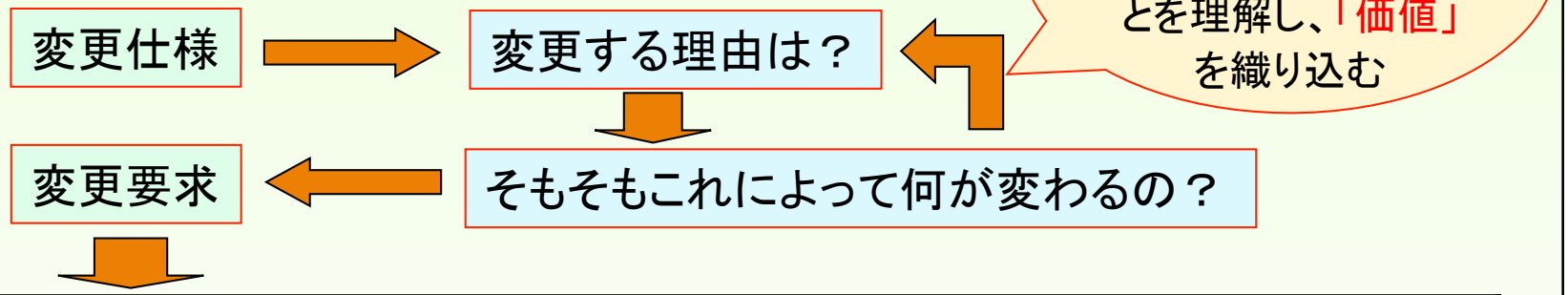
変更仕様から変更要求を立てる

- ほとんどの変更依頼は「仕様レベル」で届く

	□ □ □		防犯カメラの首振り角度を45°から60°に変更して欲しい

  - 仕様レベル・・・ソースコード上の変更箇所が特定できる状態
  - 多このまま変更すると、変更箇所が漏れることになる

- 変更仕様から変更要求を捉える



要求	撮影範囲を30%拡大し，往復時間は従来と同じにするために動作スピードをアップさせてほしい
理由	モニターも含めてトータルの設置コストを下げて販売に繋げたい

これによって、関連する変更を漏らさない

変更仕様を＜グループ＞でまとめる

- 変更仕様がいろいろな箇所に散在する場合は＜グループ＞でまとめる
- 変更箇所を見つけながら変更箇所の「TM」を作成する

				file	file	file	file
要求	CCL30	加入者データに家族データを追加して家族割りサービスを始める					
	理由	同業他社との競争に勝つため					
		＜加入者データの追加＞					
	□ □ □	CCL30-01	主従区分と10個の家族データを追加する 追加する家族データの属性は以下の通り ・加入者数 ・加入者番号		f1()		
		＜加入者データの表示の変更＞					
	□ □ □	CCL30-05	加入者名の横に主従区分の表示を追加する				f3()
	□ □ □	CCL30-06	主従区分＝主の時は、その横に家族の加入者番号を登録数分表示する			f7()	
	□ □ □	CCL30-07	主従区分＝従の時は、主となる加入者番号を表示する			f7()	
		＜計算方法の変更＞					
	□ □ □	CCL30-10	主従区分＝主に繋がる加入者の利用料金をまとめて計算する方法に変更			f8()	
	□ □ □	CCL30-11	加入者数に応じて以下の割引率の適用を追加する				f9()

機能追加を受け入れるための変更を扱う

- 追加機能を受け入れるための変更を変更要求仕様にまとめる

変更要求仕様

要求	MALX1	事前に指定された受信および送信した電子メールをキーワードで検索してメーラー上で再利用する機能を追加する	
	理由	メールが多くて,関連するメールを 探せない	
	要求	MALX1-02	検索グループを表示して選択できる機能を追加する
		理由	複数の検索グループから選択させたい
	要求	MALX1-03	キーワード入力して検索する処理を追加する
		理由	可能性のあるキーワードで探したい

[MAL01-02]の機能を受け入れるための変更を記述する

[MAL01-03]の機能を受け入れるための変更を記述する

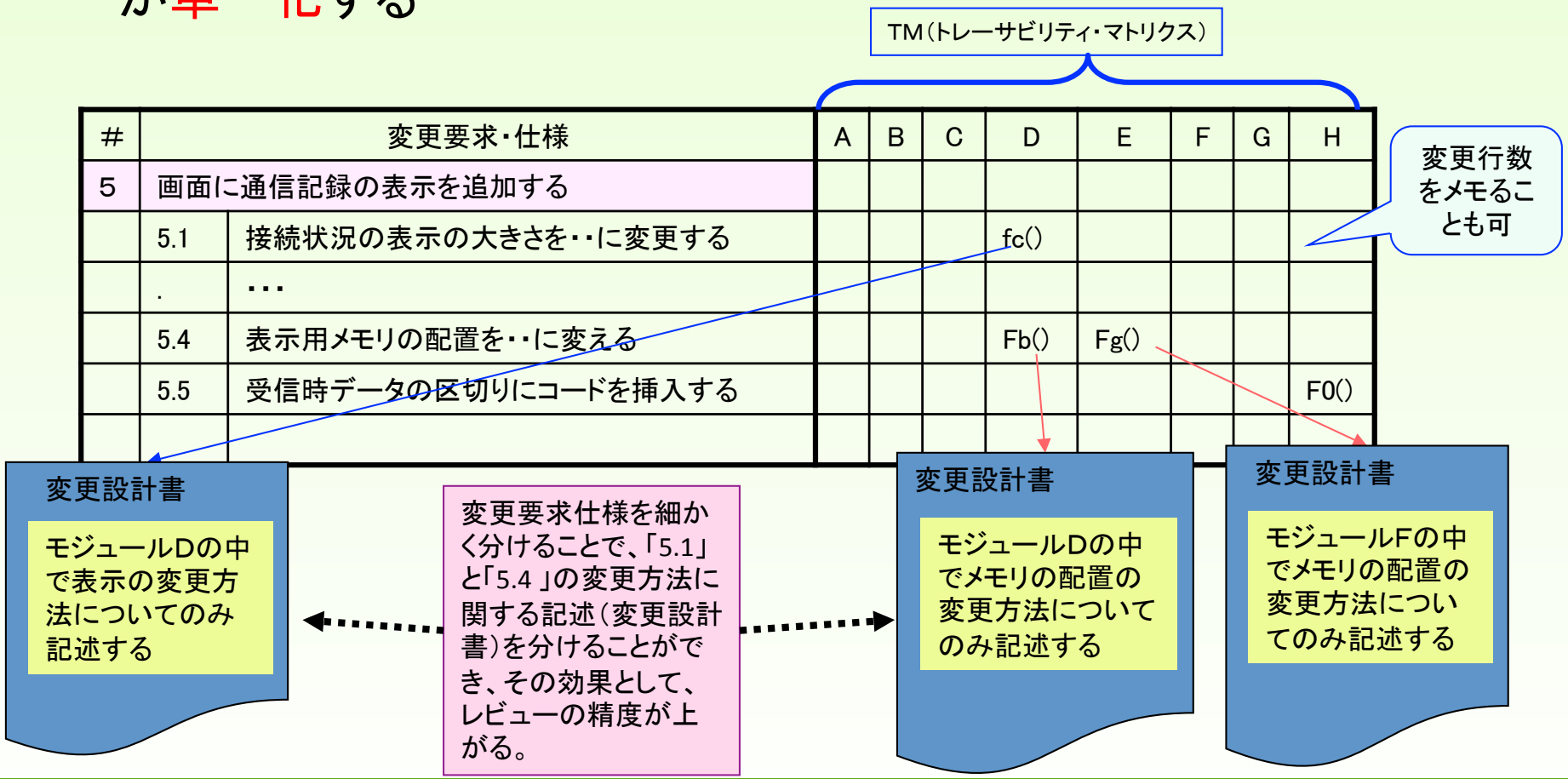
追加機能要求仕様

要求	MAL01	事前に指定された受信および送信した電子メールをキーワードで検索してメーラー上で再利用したい	
	理由	メールが多くて,関連するメールを 探せない	
	要求	MAL01-02	検索グループを一覧から指定する
		理由	検索グループが複数あるから
	□□□	MAL01-02-1	検索グループを○○データから読み出す
	□□□	MAL01-02-2	読み出したグループのデータを一覧で表示する
	要求	MAL01-03	いくつかのキーワードを組み合わせて検索できる
		理由	可能性のあるキーワードで探したい
	□□□	MAL01-03-1	検索したいキーワードを入力できる
	□□□	MAL01-03-2	複数のキーワードを「AND」と「OR」で繋ぐことができる
	□□□	MAL01-03-3	キーワードは最大8個まで指定できる

- これで追加機能が問題なく作動するかどうかを確認できる

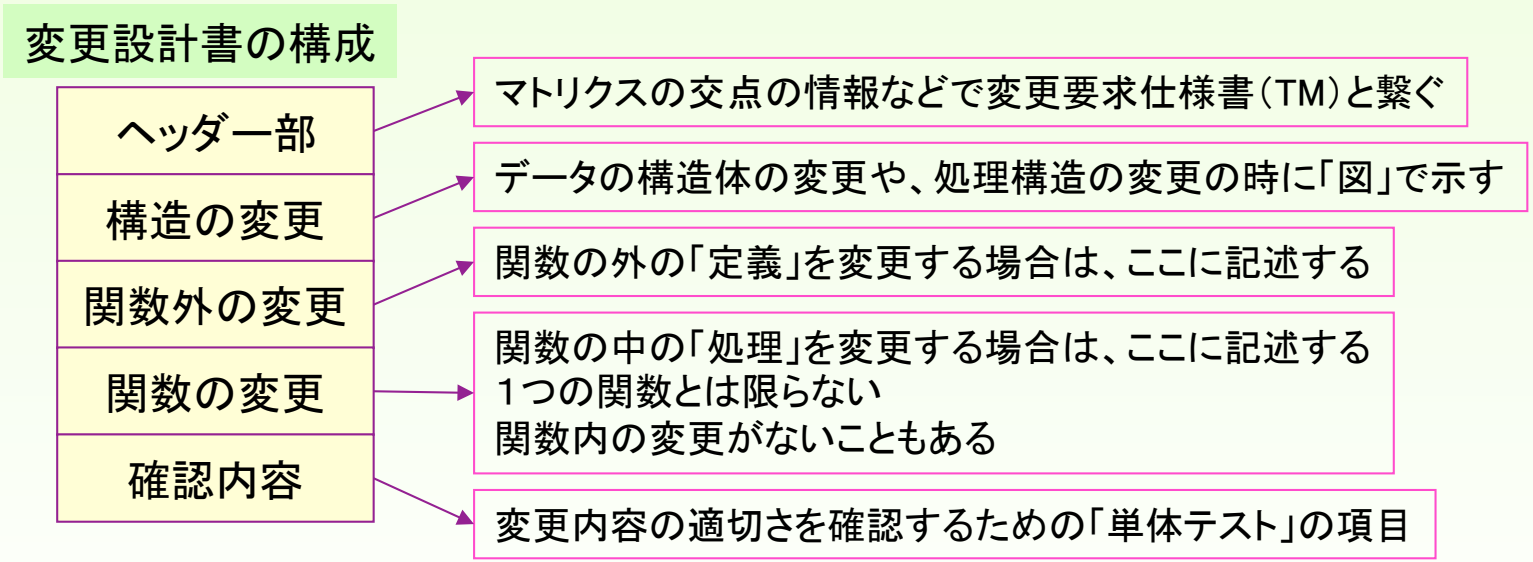
TMを介して変更設計書と繋ぐ

- 変更仕様に該当する箇所をTM(トレーサビリティ・マトリクス)上に表す
- 変更要求仕様を適切に細分化することで、変更設計書が扱うテーマが単一化する



具体的な変更方法は変更設計書で対応する

- TMに「関数名」が示された単位で変更設計書を作成する
- 変更設計書には具体的な変更方法をできるだけ文章で記述する
  - 変数や定義名はソースコードを（　）で付記することを許可する
- 変更する「差分」の記述に徹する
- 変更に伴うテスト内容（単体テストに相当）もここで記述する



一気にソースコードを変更する

- 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に必要な工数は確認されている
- 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

ソースコードの変更を見合わせて、もう一度「変更設計書」の記述を見直し、不足する情報を補ってから再開する

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

バグによる修正ヶ所を追記する

- バグが発生したときは、変更設計書や変更要求仕様書も訂正する
  - この時、「変更制御プロセス」の中で対応すること

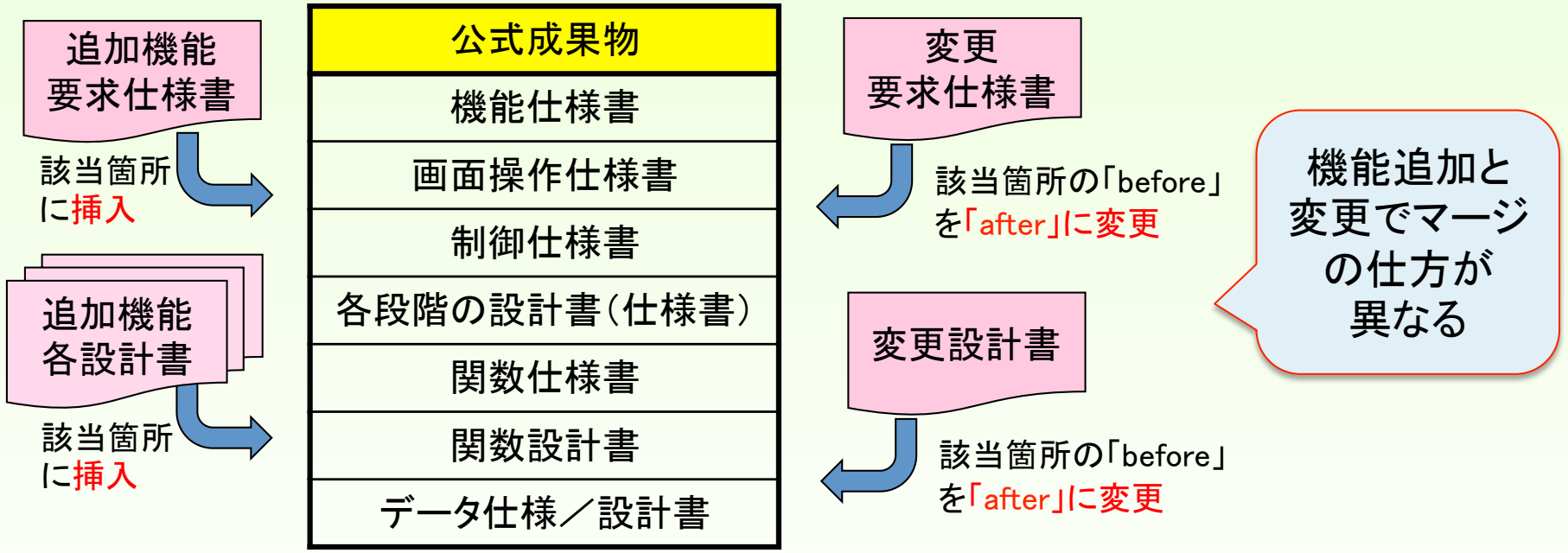
	変更要求仕様書	TM	変更設計書
変更内容（項目）がモレた	変更仕様を追加	マークを追加	変更設計書を追加
変更の必要がなかった	変更仕様を削除	（行が削除）	変更設計書を削除
		マークを削除	変更設計書を削除
			変更設計書を訂正
別のモジュールにも変更すべき箇所があった		マークを追加	変更設計書を追加
関数の中の変更方法を間違えた			変更設計書を訂正

ソースコード上の変更箇所は、すべて「変更３点セット」に記載されている状態を確保する

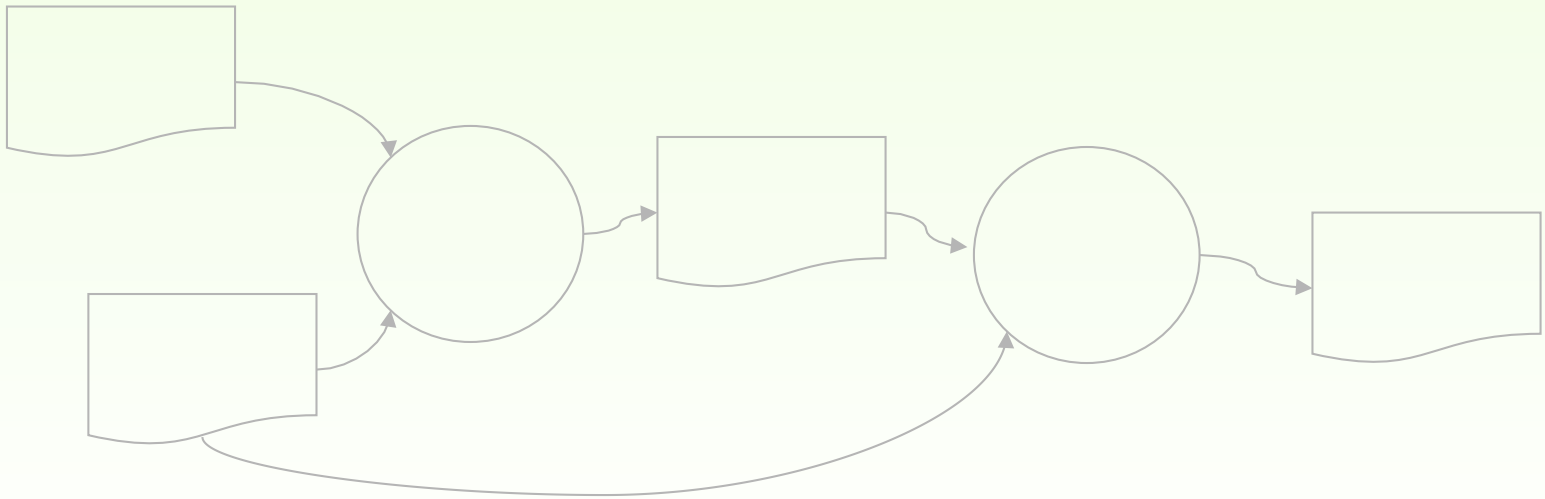
変更のエビデンスとなる

公式文書はテスト後にマージする

- ベースの「公式文書」はテスト終了後に短期間で変更する
  - テストによってその変更が正しいことを確認している
  - 更新作業は「構成管理」の手順に従う

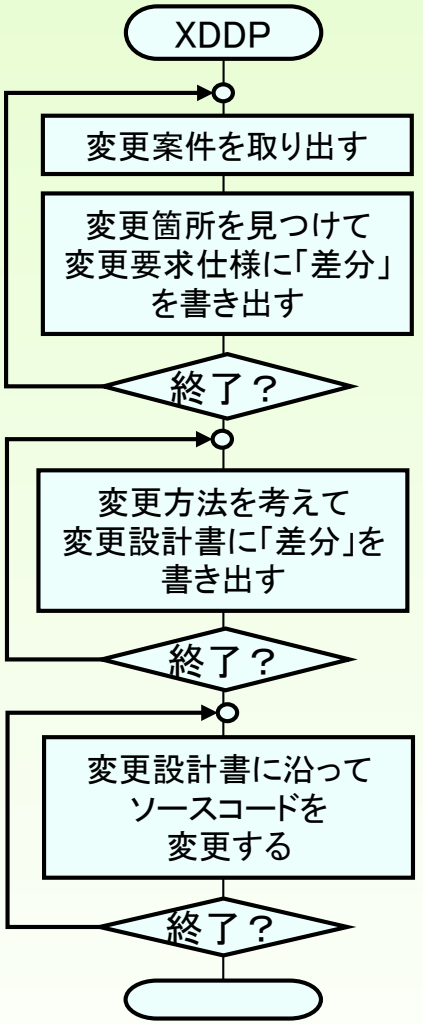


1. 派生開発の現場
2. 派生開発の特徴
3. XDDPの要点
4. XDDPとアジャイル



XDDPはウォーターフォールなの？

- 講演後によくある質問
  - 質 「XDDPってWFですか？」
  - 私 「もしかして、このフローがWFに見えるのかな？」
  - 質 「はい」
  - 私 「成果物を順次作っていって最後にソースコードを変更する形がWFだというのであれば、一般に行われている次々とソースコードを変更していく「いきなりの変更」は何？



派生開発で守らなければならないことは何か？

新規開発と派生開発の違いは何か？

機能追加と変更の違いは何か？

XDDP自身のアジャイル性

- 機能追加と変更を異なるプロセスで対応
  - 性質に見合ったプロセスで対応
- 必要最小限の成果物とプロセスの合理的な連鎖
  - 「変更3点セット」でレビューの効果を発揮
  - 最小限の秩序で派生開発特有の混乱を回避する
- ソースコードの変更を遅らせることで品質を確保
  - 派生開発では拙速なソースコードの変更は禁物
  - 複数の実現方法を検討する機会を確保

USDM→「要求」に「価値」を織り込む

リーン開発

XDDPは徹底的に無駄を省いて「QCD」を確保する方法

機能追加

- USDMの特性を活かして短時間で高精度の要求仕様が書ける
  - 「要求」と「理由」を使って、この要求の必要性や「価値」を表現できる
  - 「理由」はこの要求の仕様や設計時に配慮される

必要な機能仕様を素早く書けることで、その上に、「理由」などを考えながら、この機能の使い勝手や将来の変化の方向などを顧客と一緒に検討できる

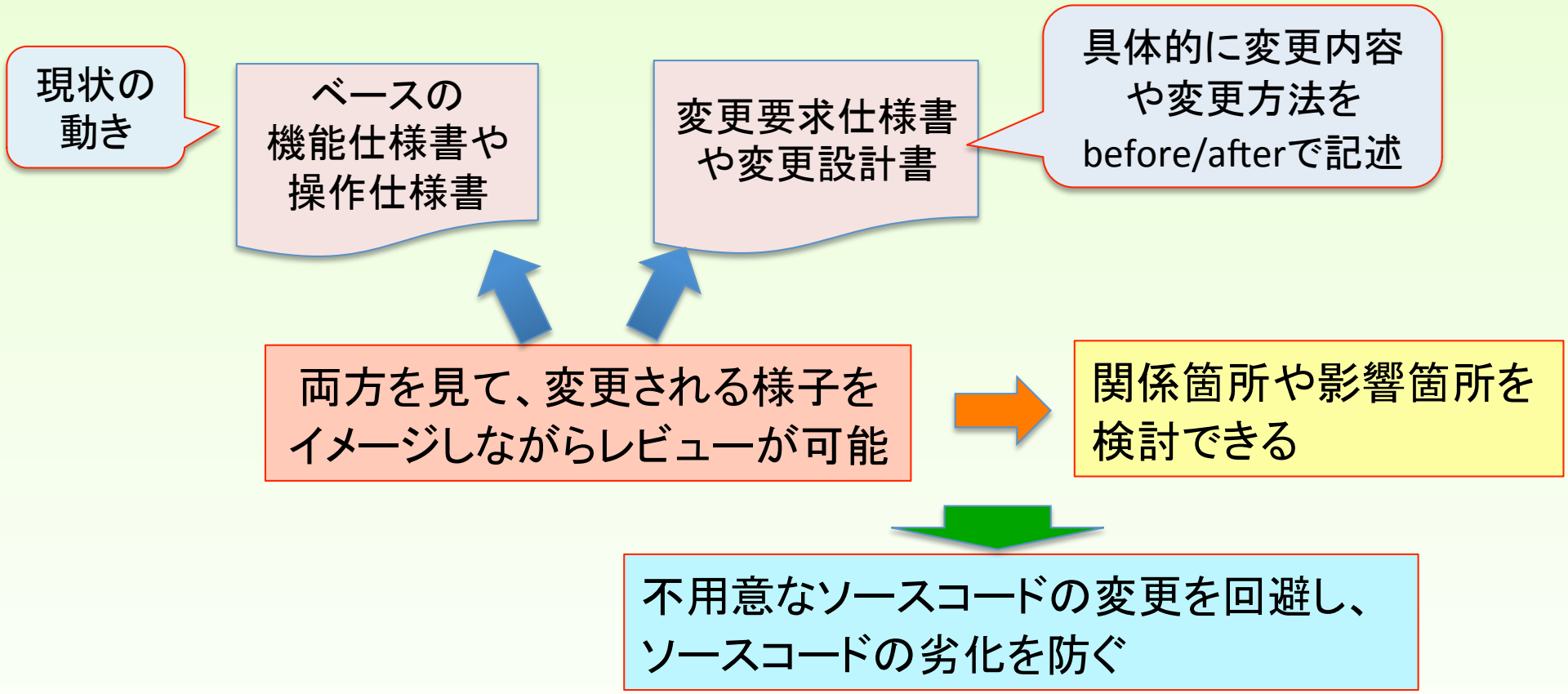
仕様が書けない状態では、レビューでは機能仕様を満たすだけに終始する

- 受入の変更点を把握したあとは設計に着手可能
  - 受け入れ方法を検討していることで影響が出にくい
  - 設計以降の実現方法はXDDPとしては関知しない
  - 追加機能単位で実装・テストまで進めることが可能

モデル駆動などに取り組むのも良いでしょう

変更

- 変更要求仕様や変更設計書には何を書いていますか？



ソフトウェア技術者を憧れの職業に！

- 今日、ビジネスの勝敗を決めるのはソフトウェア
- 製品の勝敗もソフトウェアで決まる時代に入っている
  - 製品やシステムの価値もソフトウェアで決まる
- 必要なのは「技術」
  - ソフトウェアの開発技術
  - プロジェクトマネージメント技術
  - 組織のマネージメント技術

問題は技術の不足によって起きている

これがソフトだ

コストで血眼になってオフショア先を探しているあいだに、
まともなソフトウェア技術者がいなくなれば産業は成立たない

高い技術力を身につけることで、大幅なQCDの改善が可能 ！

（参考文献）

- 文献（単行本）
  - ① 【改訂第2版】要求を仕様化する技術・表現する技術（技術評論社）
  - ② 『派生開発』を成功させるプロセス改善の技術と極意（技術評論社）
  - 「SEの仕事を楽しくしよう」（SRC）
  - 「わがSE人生に一片の悔いなし」（技術評論社：新書）

① USDM

② XDDP