派生開発で
高品質、短期間、低コスト
を実現する勘所

2014-09-05

株式会社SRA

産業第1事業部 林 好一

自己紹介： 林 好一

なりわい： ソフトウェア開発支援およびその支援

- 主な支援分野
  - ソフトウェアプロセス改善 (SPI)
  - ソフトウェアプロダクトライン (SPL) 開発
  - オブジェクト指向技術一般
- 主な支援形態
  - 調査研究、教育、定義・モデリング、メンタリング
  - 課題識別およびその解消の支援
- 最近の主な支援実績

  SPI:
  - 強み・弱み分析（韓国）、ミッションクリティカルソフトウェアプロセスの定義、海外拠点への標準プロセス展開支援、オブジェクト指向開発プロセスのメンタリング、問題解決ワークショップ

  SPL:
  - ギャップ分析～目標到達ステップの策定、ツール適用支援、顧客企業の方向性提言、SPL評価基準調査、成功事例の詳細調査、フィーチャ分析法調査、その他各種セミナー

学歴

- 早稲田大学 理工学部　物理学科 中退

参加コミュニティ

- 組込みソフトウェアギルド代表幹事、日本SPIコンソーシアム（JASPIC）運営委員、 派生開発推進協議会（AFFORDD）運営委員、ソフトウェア技術者協会（SEA）会員

コミュニティでの活動

- ワークショップ、セミナー、カンファレンス、勉強会等の企画、運営、ならびに成果の広報
- 例：
  - 「ソフトウェアプロダクトラインエンジニアリング―ソフトウェア製品系列開発の基礎と概念から技法まで」（共訳書）
  - 「XDDP と SPL の概要」（講義）
  - 「改善意識の生まれる場」（発表）
  - SPLC 産業トラック プログラム委員（貢献）

構成

- 派生開発と XDDP
  - その意味、特徴、課題
- 派生開発の課題の解消
  - 追加と変更を明確に分ける
  - 変更の影響を見極める
  - 変更の漏れ、不測の干渉をなくす
  - コード変更は全てを確認した後に行う
  - レビュー
  - サイズ見積
  - PFD
- XDDP のまとめ
- 派生開発のまとめ
- 参考：派生開発とプロダクトライン開発

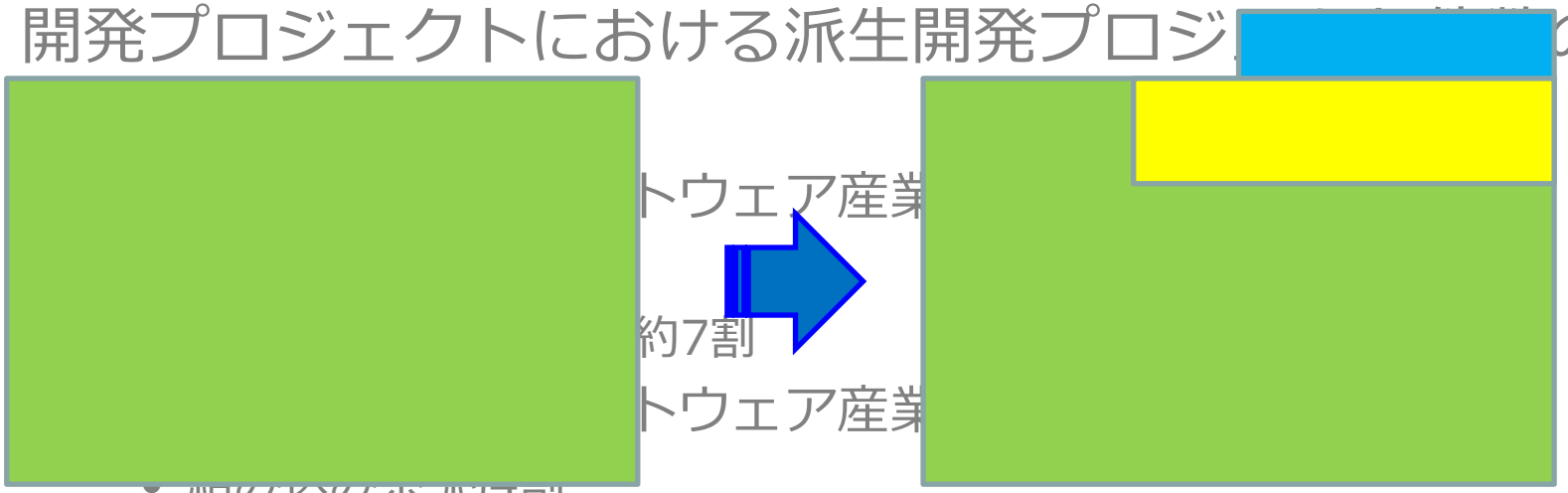
派生開発と XDDP

派生開発とは

- 派生開発とは？
  - 既にあるシステムやソフトウェアに変更・追加・削除を行い、新たなシステムやソフトウェアを作ること
  - 差分開発、流用開発、改良保守、改造開発などとも呼ばれる
- 開発プロジェクトにおける派生開発プロジェクト件数の割合
  - IPA 2011年度 「ソフトウェア産業の実態把握に関する調査」から
    - 組み込み系 約5割
    - エンタープライズ系 約7割
  - IPA 2012年度 「ソフトウェア産業の実態把握に関する調査」から
    - 組み込み系 約4割
    - エンタープライズ系 約4割
  - ただし、新規開発と分類されたプロジェクトにも、既存ソフトウェアからの部分的再利用が含まれていることから、全てが新規開発というプロジェクトの件数はさらに少ないと思われ、**8割以上が派生開発の要素を含むプロジェクト**と考えられる

派生開発とは

- 派生開発とは？
  - 既にあるシステムやソフトウェアに変更・追加・削除を行い、新たなシステムやソフトウェアを作ること
  - 差分開発、流用開発、改良保守、改造開発などとも呼ばれる
- 開発プロジェクトにおける派生開発プロジ の割



トウェア産業 から

約7割

トウェア産業 から

  - エンタープライズ系 約4割
  - 組み込み系 約4割
  - エンタープライズ系 約4割
  - ただし、新規開発と分類されたプロジェクトにも、既存ソフトウェアからの部分的再利用が含まれていることから、全てが新規開発というプロジェクトの件数はさらに少ないと思われ、**8割以上が派生開発の要素を含むプロジェクト**と考えられる

派生開発の特徴

- 変更が中心で、追加を伴うことがある
  - 稼動していたシステムに手を加えることになる
- 納期は一般的に短い
  - ベースとなるシステムがどんなに大きくても「少しの変更/追加」と思われてしまう
- 全体を理解しないままの作業となりやすい
  - 上述の短納期が理由の一つ
  - ソースコードと設計文書が乖離していることが多い
  - ベースとなるシステムを作った者がアサインできないことがある

派生開発の課題

- 変更が中心で、追加を伴うことがある
  - 稼動していたシステムに手を加えることになる
- 納期は
- 既存部分の変更も新規開発と同じやり方で進めるため、どのように変更したのか担当者の頭の中にしかない
  - ✓ どこをどう変えるのかを文書化するプロセスがない
- （変更内容は担当者しかわからないため）漏れ、誤り等はテストまで認識できない
  - ✓ 特に複数担当者による変更の間の干渉は実装後に結合して初めてわかる

派生開発の課題

- 変更が中心で、追加を伴うことがある
  - 稼動していたシステムに手を加えることになる
- 納期は一般的に短い
  - ベースとなるシステムがどんなに大きくても「少し
-
  - 変更箇所（と思しき箇所）を見つけ次第ソースコードを変更してしまい、場当たり的な対応になることがある
  - （見つけ次第のソースコード変更のため）前の変更に対して後日認識違いに気づいたり、後の変更との不整合に気づいたりして、その度にソースコードの修正が発生する
    - ✓ 前回の修正意図を思い出せない、間違う、動けばいいやなどとなり、ソースコードが劣化する
  - ソースコード変更以外の作業が省かれる傾向がある

派生開発の課題

- 変更箇所（と思しき箇所）を見つけ次第ソースコードを変更してしまい、場当たり的な対応になることがある
- （変更内容は担当者しかわからないため）漏れ、誤り等はテストまで認識できない
- の変更/追加」ことに…れてしまう
- 全体を理解しないままの作業となりやすい
  - 上述の短納期が理由の一つ
  - ソースコードと設計文書が乖離していることが多い
  - ベースとなるシステムを作った者がアサインできないことがある

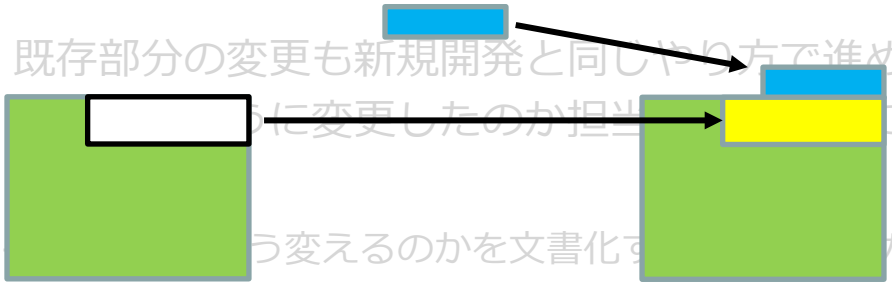
派生開発の課題の解消

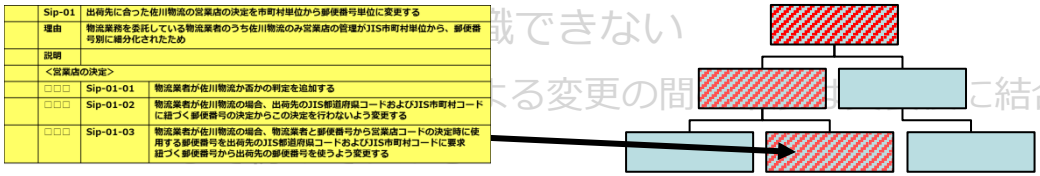
- 既存部分の変更も新規開発と同じやり方で進めるため、どのように変更したのか担当者の頭の中にしかない
  - どこをどう変えるのかを文書化するプロセスがない
- （変更内容は担当者しかわからないため）漏れ、誤り等はテストまで認識できない
  - 特に複数担当者による変更の間の干渉は実装後に結合して初めてわかる
- 変更箇所（と思しき箇所）を見つけ次第ソースコードを変更してしまい、場当たり的な対応になることがある
- （見つけ次第のソースコード変更のため）前の変更に対して後日認識違いに気づいたり、後の変更との不整合に気づいたりして、その度にソースコードの修正が発生する
  - 前回の変更意図が思い出せない、間違う、動けばいいやなどとなり、ソースコードが劣化する
- ソースコード変更以外の作業が省かれる傾向がある

- 追加と変更を明確に分ける
- 変更の影響を見極める
- 変更の漏れ、不測の干渉をなくす
- コード変更は全てを確認した後に行う

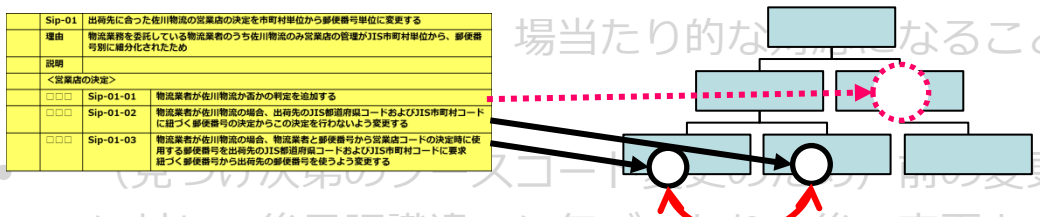
これらを確実に実行する

派生開発の課題の解消





Sip-01	出荷先に合った佐川物流の営業店の決定を市町村単位から郵便番号単位に変更する	
理由	物流業務を委託している物流業者のうち佐川物流のみ営業店の管理がJIS市町村単位から、郵便番号別に細分化されたため	
説明		
<営業店の決定>		
□□□	Sip-01-01	物流業者が佐川物流か否かの判定を追加する
□□□	Sip-01-02	物流業者が佐川物流の場合、出荷先のJIS都道府県コードおよびJIS市町村コードに紐づく郵便番号の決定からこの決定を行わないよう変更する
□□□	Sip-01-03	物流業者が佐川物流の場合、物流業者と郵便番号から営業店コードの決定時に使用する郵便番号を出荷先のJIS都道府県コードおよびJIS市町村コードに要求紐づく郵便番号から出荷先の郵便番号を使うよう変更する



Sip-01	出荷先に合った佐川物流の営業店の決定を市町村単位から郵便番号単位に変更する	
理由	物流業務を委託している物流業者のうち佐川物流のみ営業店の管理がJIS市町村単位から、郵便番号別に細分化されたため	
説明		
<営業店の決定>		
□□□	Sip-01-01	物流業者が佐川物流か否かの判定を追加する
□□□	Sip-01-02	物流業者が佐川物流の場合、出荷先のJIS都道府県コードおよびJIS市町村コードに紐づく郵便番号の決定からこの決定を行わないよう変更する
□□□	Sip-01-03	物流業者が佐川物流の場合、物流業者と郵便番号から営業店コードの決定時に使用する郵便番号を出荷先のJIS都道府県コードおよびJIS市町村コードに要求紐づく郵便番号から出荷先の郵便番号を使うよう変更する

- 追加と変更を明確に分ける
- 変更の影響を見極める
- 変更の漏れ、不測の干渉をなくす
- コード変更は全てを確認した後に行う

これらを確実に実行する

XDDP

- eXtreme Derivative Development Process
- 派生開発向けに特化したソフトウェア開発プロセス
- システムクリエイツの清水吉男が提唱している
- 前掲した派生開発の課題解消のための原則を実行するプロセスを備えている
  - 追加と変更を明確に分ける
  - 変更の影響を見極める
  - 変更の漏れ、不測の干渉をなくす
  - コード変更は全てを確認した後に行う

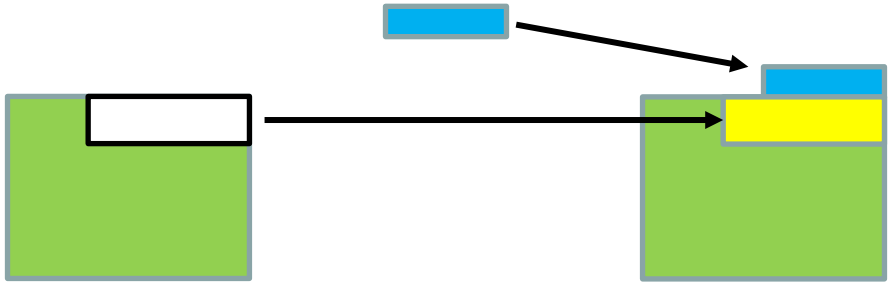
XDDPの特徴

- 部分理解を前提に、差分を扱うプロセスを提供
  - この差分を扱う変更用のプロセスとは別に追加用のプロセスもある
    - 追加用のプロセスは、機能レベルの追加がなければ使わない
- 差分を表現するための成果物を提供
  - XDDP の成果物の「3点セット」
    - 変更要求仕様書
    - トレーサビリティマトリクス
    - 変更設計書
- 部分理解の中で、「変更（と思われる）箇所をいきなり変更しない」仕組みを提供
  - レビュー（前プロセスの成果物の適切さの確認が取れたあとに次のプロセスへ進む）
  - 継続的なサイズ見積り
- 今回作るための文書とソフトウェア・機能を説明する文書を分ける仕組みを提供
- 開発プロセスを見せる・作る仕組みを提供
  - プロセス自体を Process Flow Diagram (PFD) を使って表現
    - 見えないとわからない

派生開発の課題の解消

追加と変更を明確に分ける

- 派生開発の要求項目には2種類ある
  - 追加
    - 機能レベルの追加
    - 新規開発と同じ
  - 変更
    - 削除もある
    - 仕様レベルの追加は、変更として扱う
    - 追加を動かすための変更もある
    - ベースとなるソフトウェアとの差分を扱う



- 変更と追加を区別する理由
  - 変更は既存システムとの差分
  - つまり変更要求に対して「この箇所の○○を××へ、このように変更する」を示すことが重要

追加時への対応

- 基本的に新規開発と同じ
  - ただし追加機能を呼び出すモジュールには手を加える

XDDP では…

- 追加機能要求仕様書を書く
  - 追加機能自体の要求仕様を全て記述する
  - 追加機能（新規開発）の設計への主要なインプットとして使われる
  - 一部は変更用のプロセスへのインプットとなる
    - 追加機能を既存システムで動作させるための変更は既存コードに手を入れるので「変更」として扱う
  - USDMで表現する
- 設計以降の工程は、それを実施するプロジェクトのやり方に従う

変更への対応

- 既存システムの**何をどう変えるのかを明らかに**し、レビューし、変える

XDDP では…

- 3点セットと呼ばれる3種類の成果物
  - 変更要求仕様書
    - 何から何へ変更するのか（Whatを表現する）
    - 変更前と変更後をセットで表現する
    - USDMで表現する
  - トレーサビリティマトリクス
    - どこを変更するのか（Whereを表現する）
    - 今回の変更箇所がどのソースファイルにあるかを表現する
      - ソースファイル上の該当箇所の行数と関数名を記入すると良い
  - 変更設計書
    - どのように変更するのか（Howを表現する）
    - あとで変更設計書をみたときに、変更内容がすぐ想起できるレベルで表現する

USDM による変更要求仕様の一例

ベースの要求仕様

要求	Sip-01	出荷先に合った物流業者の営業店を決定する	
	理由		
	説明		
	＜営業店の決定＞		
	□□□	Sip-01-01	出荷先のJIS都道府県コードおよびJIS市町村コードに紐づく郵便番号を決定する
	□□□	Sip-01-02	物流業者と郵便番号から営業店コードを決定する 【説明】郵便番号は直前で決定した郵便番号を使用する

変更要求仕様

要求	Sip-01	出荷先に合った佐川物流の営業店の決定を市町村単位から郵便番号単位に変更する	
	理由	物流業務を委託している物流業者のうち佐川物流のみ営業店の管理がJIS市町村単位から、郵便番号別に細分化されたため	
	説明		
	＜営業店の決定＞		
	□□□	Sip-01-01	物流業者が佐川物流か否かの判定を追加する
	□□□	Sip-01-02	物流業者が佐川物流の場合、出荷先のJIS都道府県コードおよびJIS市町村コードに紐づく郵便番号の決定からこの決定を行わないよう変更する
	□□□	Sip-01-03	物流業者が佐川物流の場合、物流業者と郵便番号から営業店コードの決定時に使用する郵便番号を出荷先のJIS都道府県コードおよびJIS市町村コードに紐づく郵便番号から出荷先の郵便番号を使うよう変更する

Before

After

仕様レベルの追加

変更仕様

Before

After

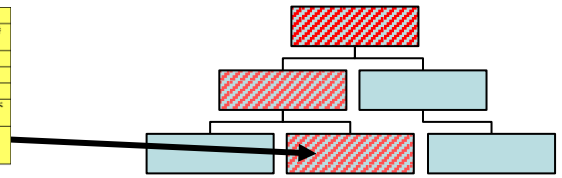
変更仕様

変更の影響を見極める

- 変更を施すとその影響が他に及ぶことがある
  - 影響に応じたさらなる変更を施すことの必要性を判断
  - 必要な場合はその変更の内容を判断
  - これを再帰的に実施

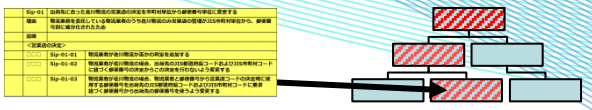
XDDP では…

	Sip-01	出荷先に合った佐川物流の営業店の決定を市町村単位から郵便番号単位に変更する	
	理由	物流業務を委託している物流業者のうち佐川物流のみ営業店の管理がJIS市町村単位から、郵便番号別に細分化されたため	
	説明		
	<営業店の決定>		
	□□□	Sip-01-01	物流業者が佐川物流か否かの判定を追加する
	□□□	Sip-01-02	物流業者が佐川物流の場合、出荷先のJIS都道府県コードおよびJIS市町村コードに紐づく郵便番号の決定からこの決定を行わないよう変更する
	□□□	Sip-01-03	物流業者が佐川物流の場合、物流業者と郵便番号から営業店コードの決定時に使用する郵便番号を出荷先のJIS都道府県コードおよびJIS市町村コードに要求紐づく郵便番号から出荷先の郵便番号を使うよう変更する



- 変更の影響を見極める仕組みが三つ
  1. 変更前（before）と変更後（after）の要求仕様をセットで表現する
     - 変更要求仕様書に変更前と変更後をセットで表現する
       - 変更前後が書かれることにより、確認レビューが活発になり、関係者への変更に関しての気づきを促す
       - 「変更前」が表現されることによる、他の影響箇所の気づきのきっかけになる（他に影響をうける箇所がないか ということを考えるきっかけ）

変更の影響を見極める

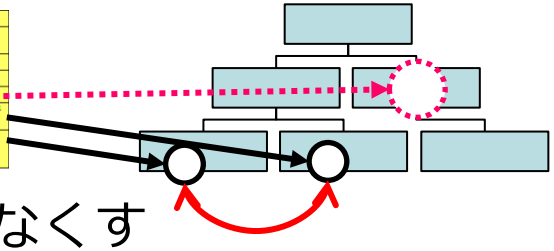


2. 変更仕様や変更内容は文書で表現する
   - 変更要求仕様書、変更設計書に表現する
     - 担当者の変更の意図がソースコードで書かれたときより確認しやすい
     - ソースコードで書いた場合、あとからその変更の意図を思い出せない。思い出せても時間がかかる
     - 一度書かれたソースコードは、より良い方法が見つかっても捨てきれない（結局直さない→ソースコードが劣化する）
3. 変更箇所が具体的にどのソースにあるかを表現する
   - トレーサビリティマトリクスにより、変更を加える箇所を表現する
     - どこを変更するかが実装前に、関係者にもわかる。干渉する箇所がわかる
     - 関係者に変更に関しての気づきを促す
       - » 同じ変更すべき箇所が別にもあるのではないか、この箇所は関係がないのではないか 等

変更の漏れ、不測の干渉をなくす

- ある要求/仕様を満たすために…
  - 必要な変更を全て網羅する
  - それらの変更によって変更する必要のある箇所の変更も全て網羅する
  - それらの変更は互いに干渉しないことを確認する

	Sip-01	出荷先に合った佐川物流の営業店の決定を市町村単位から郵便番号単位に変更する	
	理由	物流業務を委託している物流業者のうち佐川物流のみ営業店の管理がJIS市町村単位から、郵便番号別に細分化されたため	
	説明		
	<営業店の決定>		
	□□□	Sip-01-01	物流業者が佐川物流か否かの判定を追加する
	□□□	Sip-01-02	物流業者が佐川物流の場合、出荷先のJIS都道府県コードおよびJIS市町村コードに紐づく郵便番号の決定からこの決定を行わないよう変更する
	□□□	Sip-01-03	物流業者が佐川物流の場合、物流業者と郵便番号から営業店コードの決定時に使用する郵便番号を出荷先のJIS都道府県コードおよびJIS市町村コードに要求紐づく郵便番号から出荷先の郵便番号を使うよう変更する

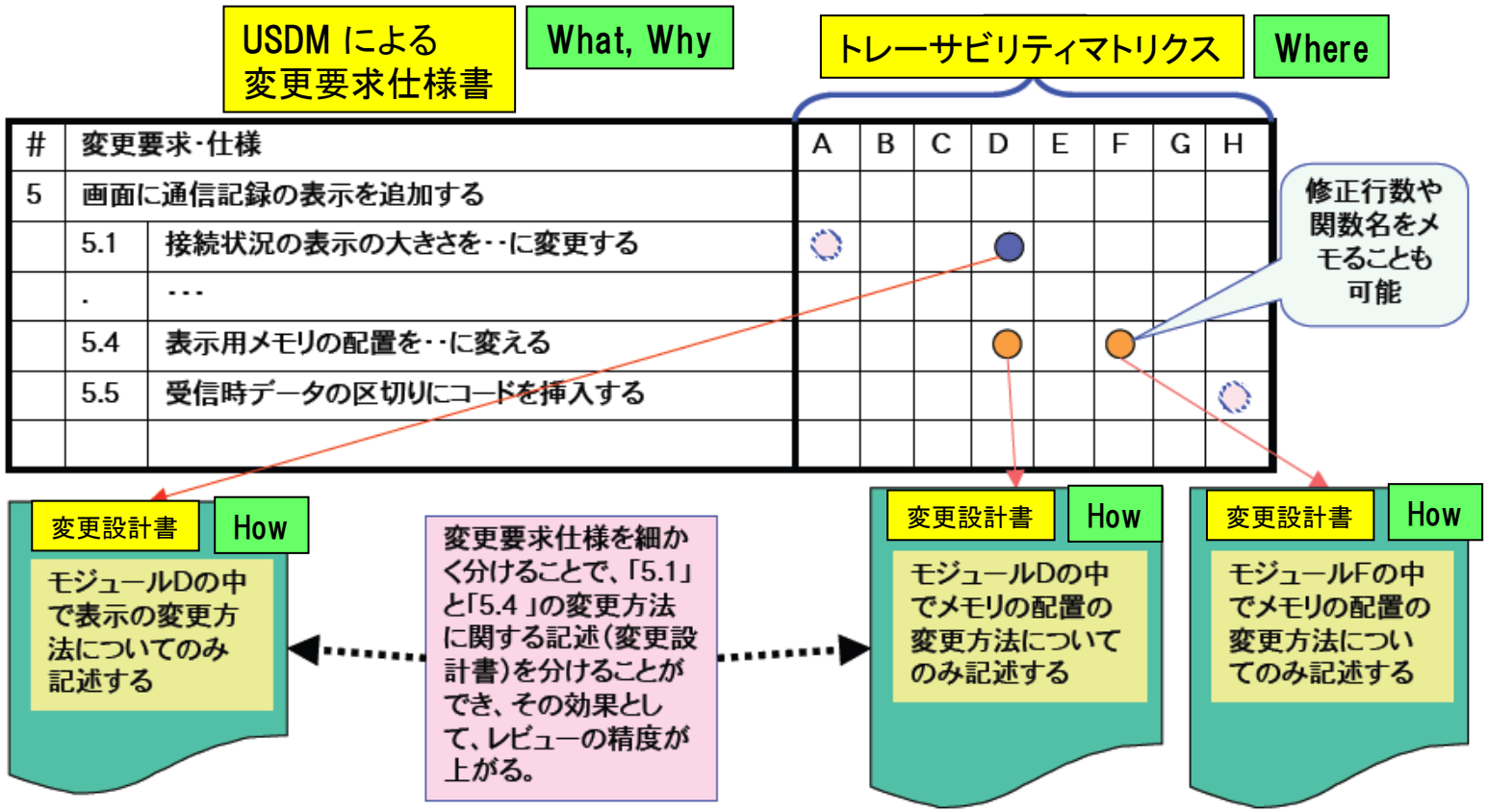


XDDP では…

- 3点セットで実装前に仕様間・担当間の干渉をなくす
  - 3点セットから変更箇所の情報によって、実装前に干渉している箇所と内容が具体的にわかる
    - トレーサビリティマトリックスを作成した時点で同一ソースコードファイルに複数交点が見つかることでわかる
  - 見つかった時点で、実装前に調整できる
    - 見つけ次第の修正で干渉がわかったら
      - こわくて修正できない。よって変更箇所をコピペし、呼び出しもとで分岐して対応する。ソースコードが劣化する
      - 後の新たな派生開発では、この変更の意図がわからない、思い出せない
      - 直接修正したら前の変更が意図通りにならない（可能性がある）
      - 前の変更内容がドキュメントに反映されていた場合、ソースコードと乖離する（可能性がある）

3点セットの成果物間の関係

- ３点セットの成果物は下図のように関係し合っている



コード変更は全てを確認した後に行う

- コードを変更することによる悪影響がないことを確認する前に行う変更には、手戻りのリスクがある
- 見逃した変更を後から行うことは、その変更による影響もまた見逃すリスクを伴う

XDDP では…

- 変更設計書に書かれている変更内容が適切で干渉し合わない確認がとれた後に、コードを変更する
  - 3点セットの作成とこれらのレビューにより、十分仕様が調整され、干渉もなくなっている
    - 実装時に悩む、調整することが少ない
  - 実装までに何度もソースコードを読んでおり、今回の変更に関する既存ソースコードの理解は、プロジェクト当初よりも理解が数段進んでいる
    - 実装時に悩む、疑問に思うことが少ない

部分理解による間違いを防ぐ（レビュー）

- 部分理解の中で、見つけ次第のソースコードの変更を防ぎ、また勘違い、思い込みによる間違いや他への影響があるかを確認し、モレを防ぐための１つの仕組み
  - 自分で抽出した変更情報の勘違いや思い込みは自分ではなかなかわからない（担当者は正しいと思うから抽出した）
    - レビューなどの第3者の目を入れることが必要
- 3点セットでWhat – Where - How とbefore/after が具体的に記述されるので、気づきを促す効果が高い

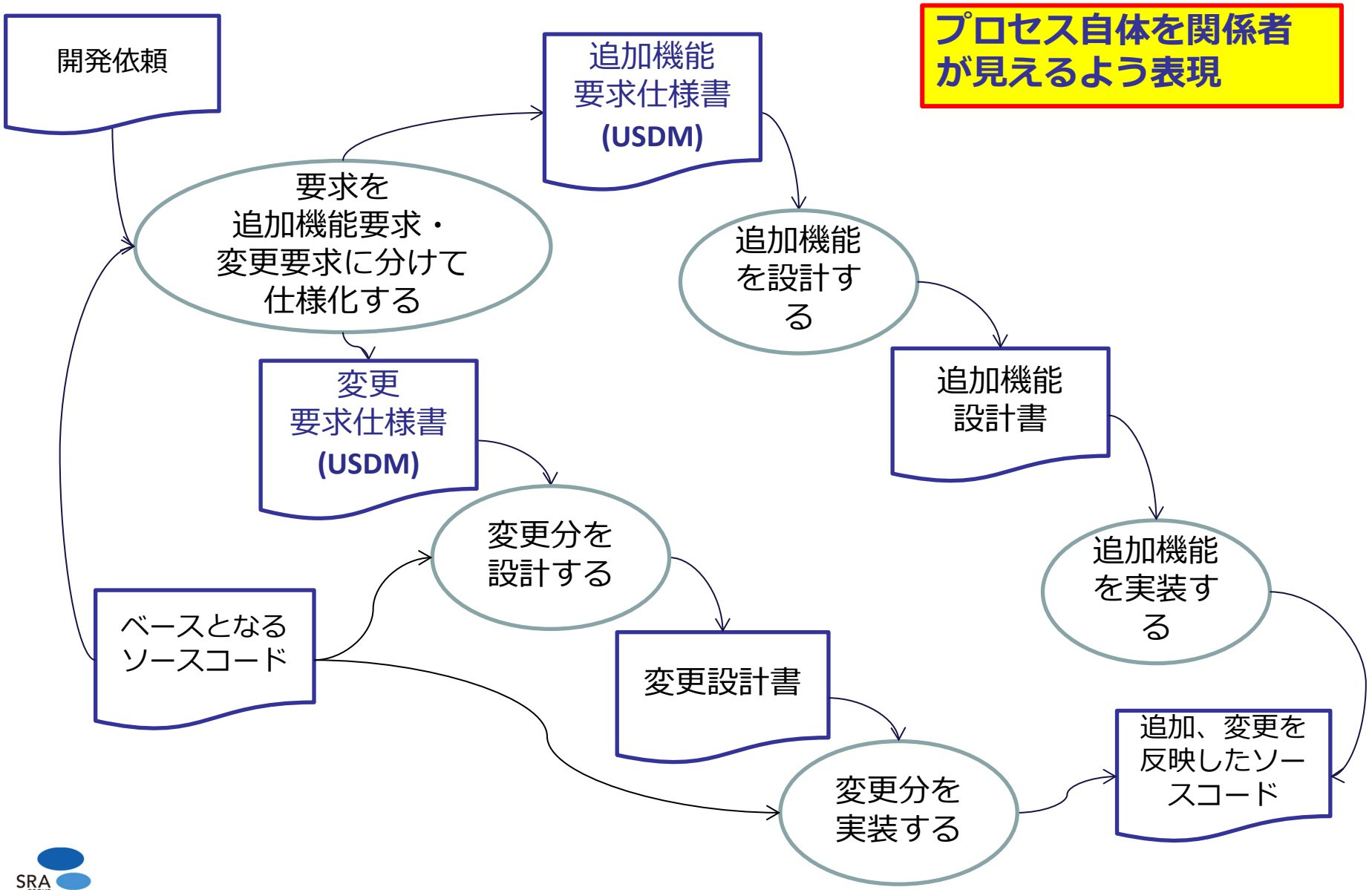
レビューの視点	レビュー対象	タイミング	レビューポイント
何を変更するのか（What)	変更要求仕様書	・既存仕様書から仕様を抽出した後 ・ソースコードから仕様を抽出した後（変更設計前）	• ここを変更すればよいという変更箇所(仕様)の認識が適切か
どこを変更するのか（Where）	TM	・TM作成後（変更設計前）	• 無関係な場所ではないか（見つけ過ぎ） • 他に変更する箇所はないか（漏れ）
どのように変更するの（How）	変更設計書	・変更設計作成後（実装前）	• 変更仕様で捉えられている変更の意味とマッチしているか • 具体的な変更方法が適切か

見つけ次第の変更の抑制（サイズ見積り）

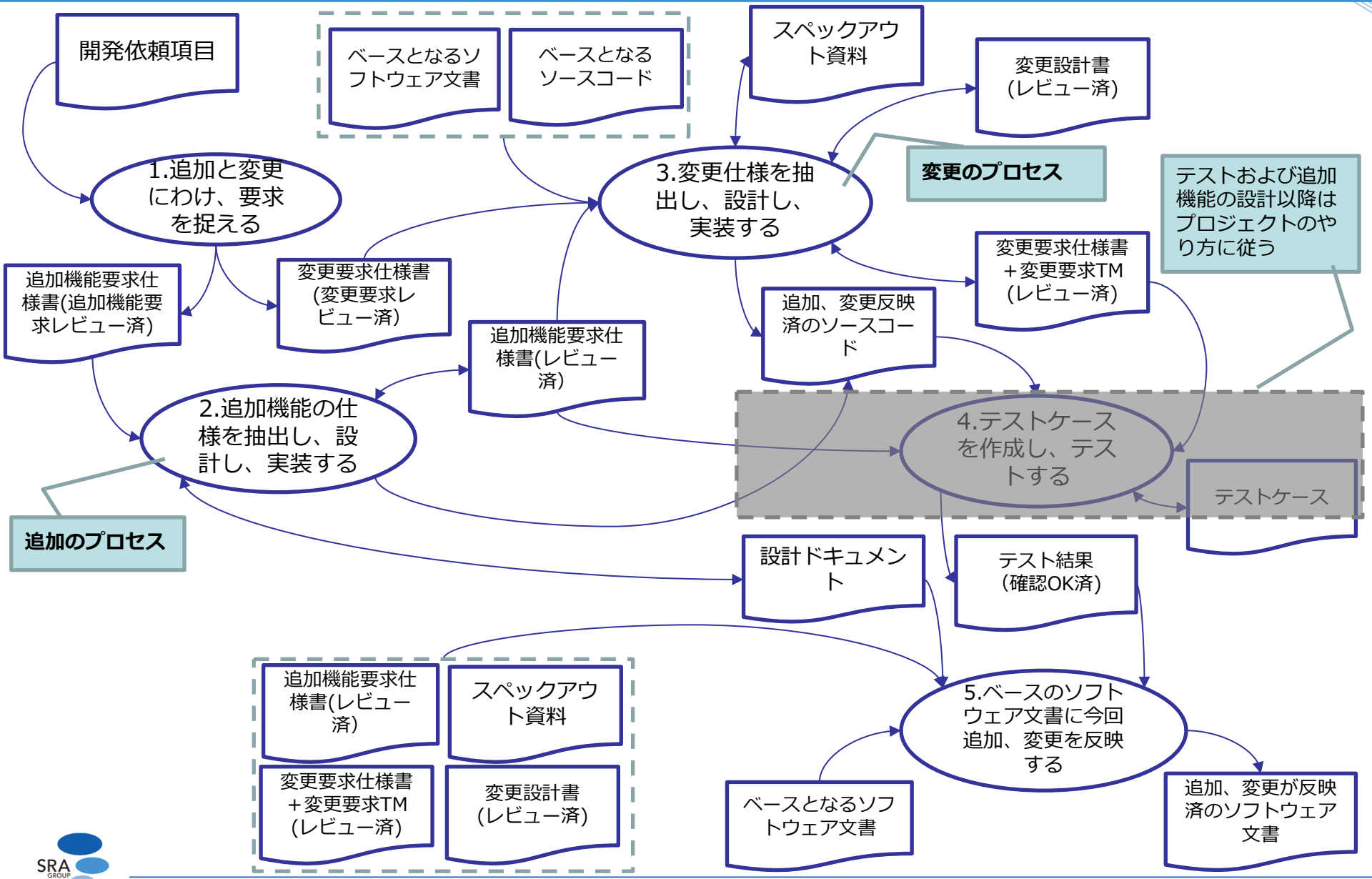
- サイズ見積り
  - 変更仕様項目数と変更するソースコードの行数を見積る
  - ここから実装工程にかけられる時間、工数を把握する
  - 継続的に実施する
- ソースコード変更行数を見積らないから、間に合わない不安になり、いきなり修正してしまう。これを防ぐ１つの仕組み
- 実装にかかる時間を把握し（実装以前に使える時間を把握する）、ぎりぎりまでソースコードを変更させず、勘違い、思い違いの排除とより良い変更方法を発見することを3点セットを使って、実施したい
- サイズ見積りでは、“前回の見積り時より超えていないか”、“超えていても対応できる範囲か” を確認する
  - 対応できない場合、不足分の工数を確保するために人員増加、トレードオフの調整へ

タイミング	仕様項目数	変更行数
変更要求記入後（仕様抽出前）・・仮説見積りと呼ぶ	初期の見積り値	初期の見積り値
変更要求仕様書記述後	実績値	仕様の実績値から再見積り
TM記入時	-	ソースコード見ながら再見積り
変更設計書記入時	-	ソースコード見ながら再見積り
ソースコード修正	-	実績値

PFD ― プロセスの設計・再設計のため



PFD による XDDP の表現

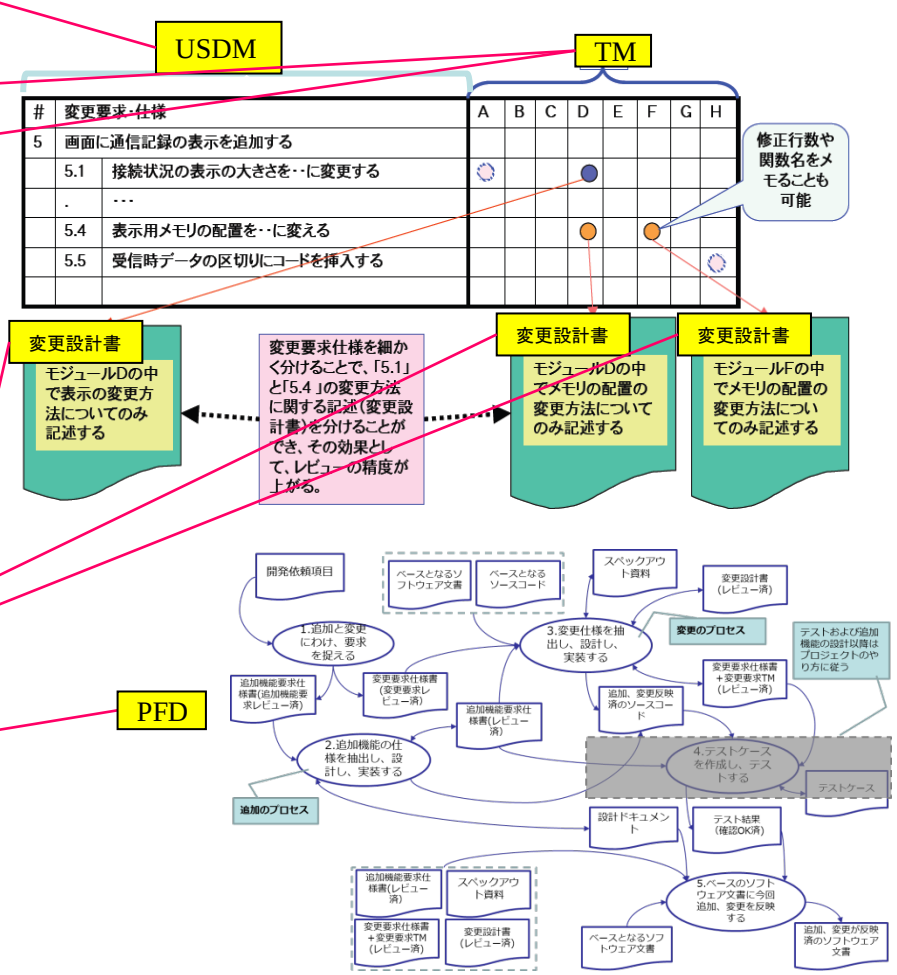


XDDP のまとめ

手戻りを起こさず最初から正しく作るために以下が用意されている

- 仕様の抜け・漏れをなくす
- 追加と変更を明確に区別する
  - 追加：
    - 単に新たに作ればよい
- 変更：
  - ひとつの仕様に対して複数ありうる
  - 複数の仕様が同じ箇所の変更で実現されることがありうる

  そのために…
  - 変更はその箇所と内容を全て明らかにしてからコーディングする
- そしてこれらのプロセスを状況に応じて PFD で設計・再設計する



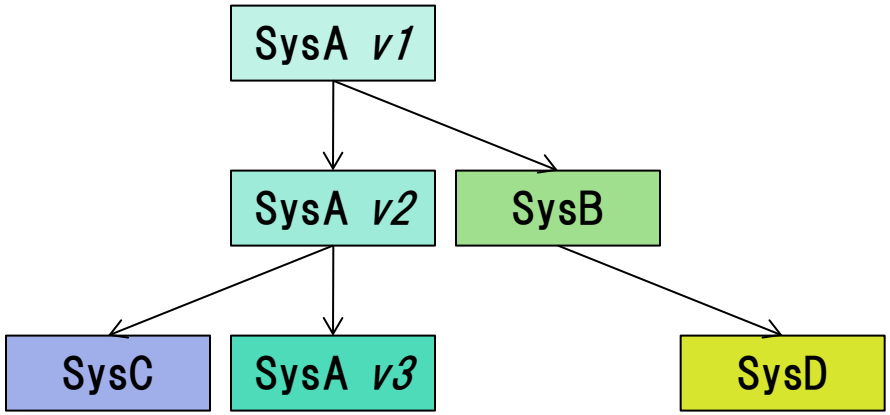
派生開発のまとめ

- 時間を節約しようとして必要な作業を端折るとかえって作業が増える
  - 変更の内容、場所、方法の明確化が必要
- 端折ると品質も低下する
  - 誤りを招き、それに気づかずに作業を重ねる
  - コードが劣化する
- 高品質、短期間、低コストを達成するためには必要なことを省かないこと
  - 追加と変更を明確に分ける
  - 変更の影響を見極める
  - 変更の漏れ、不測の干渉をなくす
  - コード変更は全てを確認した後に行う

参考：派生開発とプロダクトライン開発

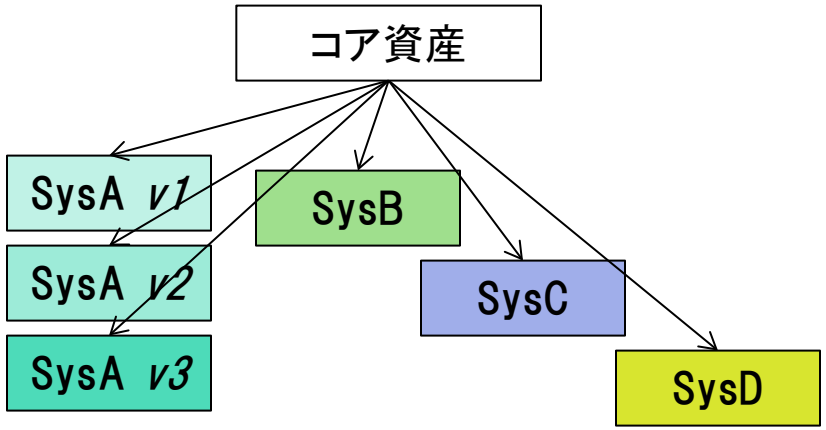
どちらも再利用による効率向上が狙い

派生開発の典型例



- 過去のシステムを基に再利用の形を検討する
  - 作られているものを再利用する

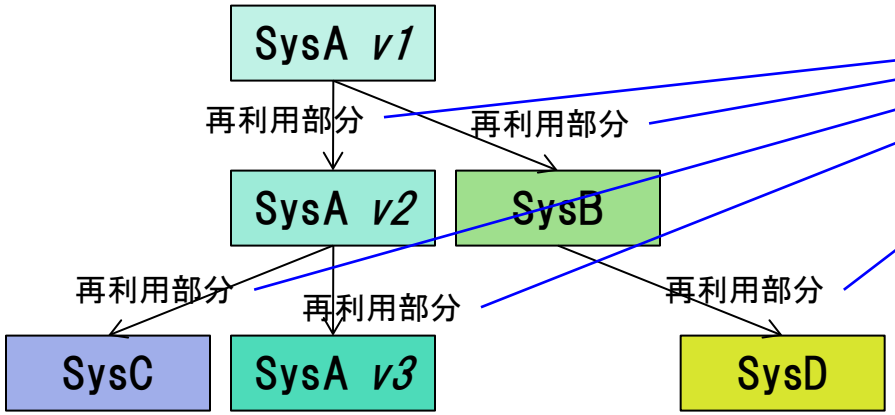
プロダクトライン開発の基本形



- 将来のシステムを基に再利用の形を検討する
  - 再利用するものを用意する

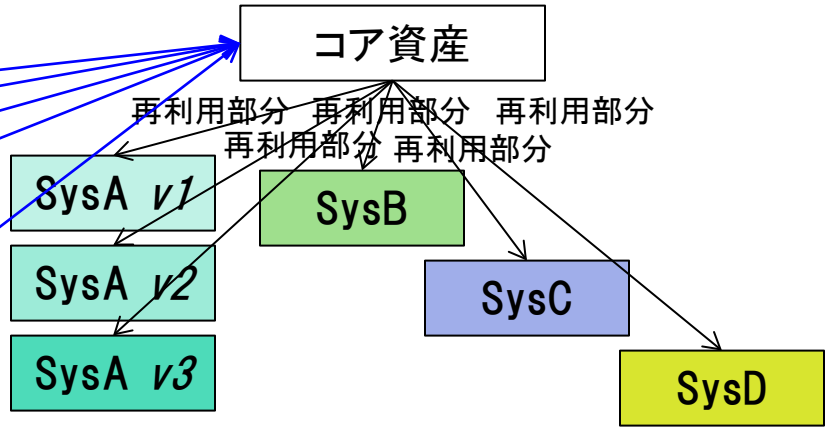
参考：派生開発からプロダクトライン開発へ

派生開発



- 過去のシステムを基に再利用の形を検討する
  - 作られているものを再利用する

プロダクトライン開発



- 将来のシステムを基に再利用の形を検討する
  - 再利用するものを用意する

参考資料

- 書籍「派生開発」を成功させるプロセス改善の技術と極意
  - 著者：清水 吉男
  - 出版社：技術評論社
- IPA 独立行政法人 情報処理推進機構
  - 2011年度 「ソフトウェア産業の実態把握に関する調査」
  - 2012年度 「ソフトウェア産業の実態把握に関する調査」
- 派生開発推進協議会ホームページ
  - http://www.xddp.jp/