そうなんだ！XDDP

派生開発カンファレンス 2013
チュートリアル資料

(株) デンソー
プロジェクトマネージャー
古畑 慶次
kkobata@ndtec.denso.co.jp

Confidential

自己紹介

古畑 慶次　（こばた けいじ）

＜所属＞　：　(株) デンソー技研センター

プロジェクト・マネージャー / 産業カウンセラー

＜業務＞　：　技術支援・指導、研修企画、研修講師

- 派生開発、プロセス設計、仕様化技術の技術支援・指導
- 高度技術者（トップガン）研修の企画、運営
- 産業カウンセラーとしてカウンセリングも行う

＜略歴＞

1988年　：　(株) デンソー 入社　（日本電装株式会社）
研究開発部　・・・　デジタル信号処理

1990年　：　基礎研究所　・・・　音声認識

1994年　：　通信技術部　・・・　ソフトウェア設計 / ハードウェア設計
（携帯電話 / PHS 親機）

2002年　：　ITS 技術部　・・・　プロセス改善　（CMM / 現場改善）

2004年　：　デンソー技研センター　＜現在に至る＞

DENSO

3つの質問

質問１　：　派生開発は新規開発よりもやさしい？

- 新人を投入するのは、いつも派生開発のテスト　・・・
- 参考になるソースコードがあるから　・・・

質問２　：　ドキュメントがないからXDDPは適用できない？

- 設計書がないから難しいんじゃないの　・・・
- ソースコードしか残っていないし　・・・

質問３　：　一人ではXDDPに取り組むことはできない？

- だって、標準プロセスがあるし　・・・
- みんなで全部のプロセスを入れ替えないと　・・・

午後のワークショップ

- テーマ１ ：
  XDDP入門の難所を考える
- テーマ２ ：
  導入提案時の障壁の克服方法
- テーマ３ ：
  PFDの賢い使い方、活用
- テーマ４ ：
  スペックアウトのコツと勘所

チュートリアルの内容

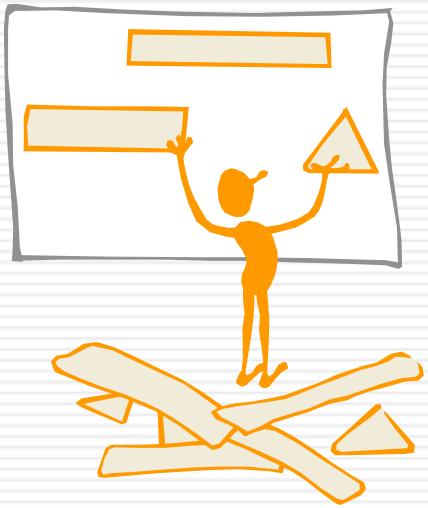
Agenda

1. 派生開発って簡単なの？

2. XDDP の「超」入門

3. XDDP の 勘所！

4. いつやるか？ 今でしょ！



1．派生開発って簡単なの？

- 派生開発とは何か？
- 派生開発の現実
- 要求の多様性
- プロセスの問題
- 部分理解
- 派生開発の難しさ
  - 開発プロセス
  - 影響範囲の特定

DENSO

1.1 派生開発とは何か？

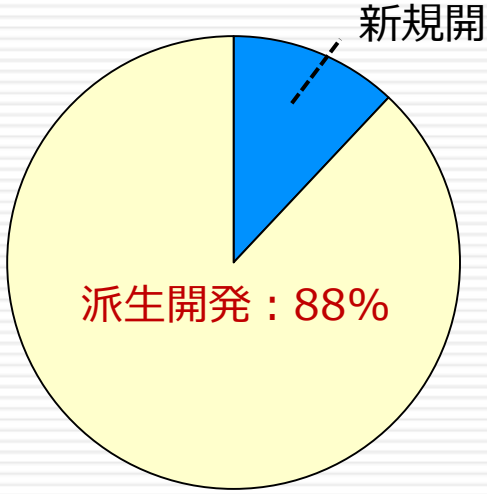
- 「派生開発」 ： 新規開発に対峙させた概念
  - 従来製品（プログラムコード）に、製品の価値を高めるために新しい機能を追加したり、これまでの機能を改善する開発
- 保守開発 ［JIS X 0161］［ISO14764］（http://www.jisc.go.jp/ ）
  - ソースコードの修正／変更は、ソフトウェア・エンジニアリングでは「保守」あるいは「保守プロセス」として扱われている
    - 是正保守、予防保守、緊急保守、適応保守、完全化保守、改良保守
- 派生開発 vs 保守開発
  - 組込みシステムの開発は「保守」では説明がつかない

「派生開発」 ＝ 保守開発 ＋ （機能追加＋仕様変更＋バグ修正）

1.2 ソフトウェア開発の現状

- ソフトウェア開発の大半は 「派生開発」

開発
- 新規開発
- 派生開発



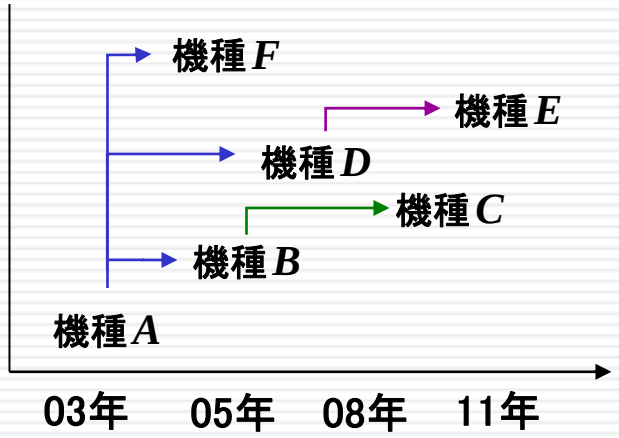
＜ 2009年度 開発行数 ＞

出典： 2009年組込みソフトウェア産業実態調査（IPA）

派生開発とは？

既存製品への機能追加や
機能の変更・削除による製品開発

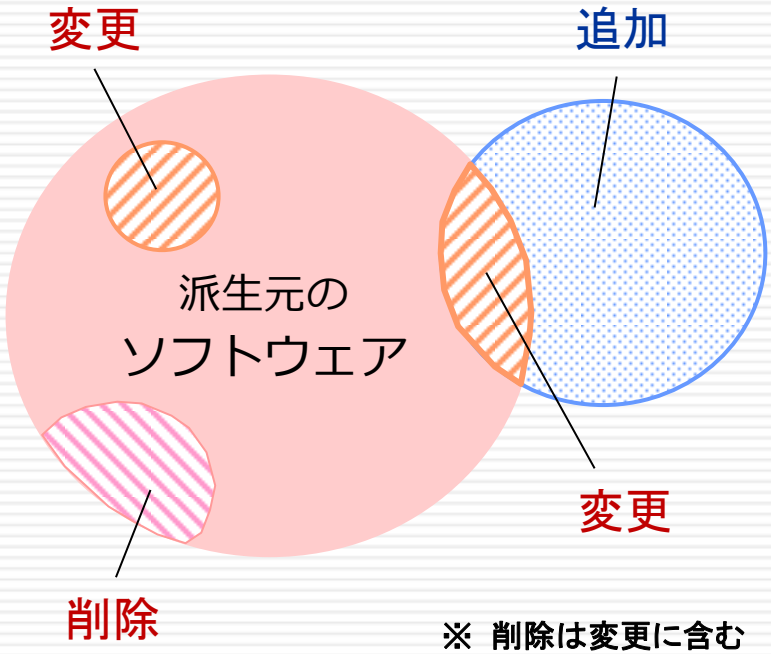
例）初期の携帯電話 → 今日のケータイ端末



＜ 製品展開 ＞

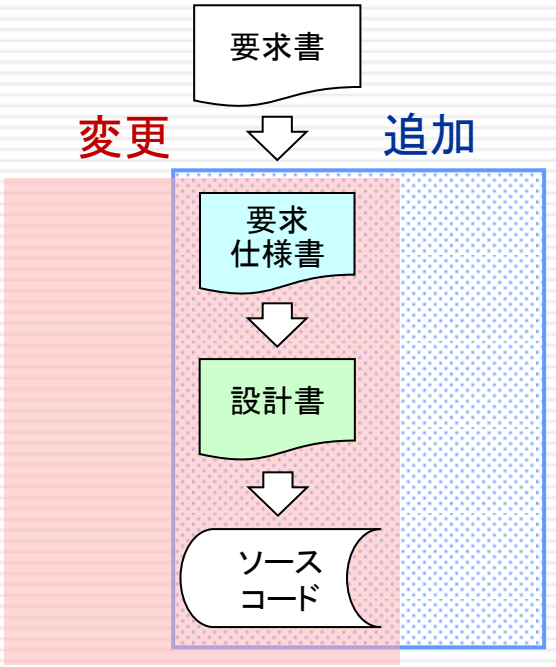
1.3 派生開発の現実

- 派生開発での開発項目
  - 「追加」と「変更」の開発が混在



➡ 要求の多様性 への対応

- 開発プロセス
  - 新規開発のプロセスで対応



➡ プロセスの問題

1.4 要求の多様性

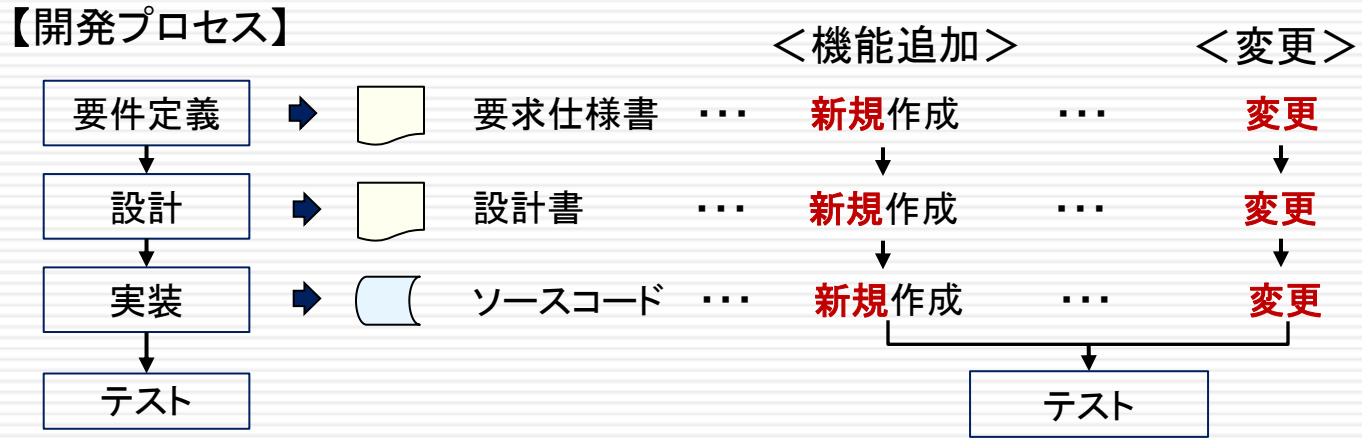
- 派生開発の要求は多種多様
  - 機能 ・・・ 追加、削除、移植
  - 仕様 ・・・ 変更、追加、削除
  - 規模 ・・・ 小規模 ～ 大規模 （様々）
  - 制約 ・・・ 納期、コスト、要員、作業

⇒ 合理的なプロセス、成果物が求められる

- 要求への対応の難しさ
  - ソースコードの状態や担当者の知見レベルの差
  - 変更量によっては作業を分割 ← 情報共有の問題
  - 要件によって完成状態を定義する必要も生じる

派生開発では多種多様な要求に対応しなければならない

- 新規開発と同じプロセスで開発
  - 仕様書から設計ドキュメント、ソースコードを順に作成（変更）する



    - 変更と機能追加は、要求が異なるにもかかわらず同じプロセス

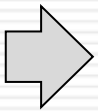
  - 新規開発のプロセスでは限界がある
    - 変更点の影響範囲や関連箇所に気づくにくく、変更モレや間違った変更が生じやすい

「部分理解」

    - 時間切れとなり、途中のプロセスを飛ばすケースも ・・・

1.6 部分理解の罠

- 反省会での発言：
  - 「ソースコード全体を理解できなかったことが問題」
  - 「時間が足りなかったので、もっと早くソースコードに手をつけます」
  - これで本当によくなるでしょうか？
- ソースコードはどうなっているのでしょう？
  - 保守性を無視して作られている
  - これまでの変更で構造が崩れている
  - 理解の参考になる成果物はない
  - 担当者の読解技術も時間も不足



全体を理解できる
状況ではない

派生開発は「部分理解」の制約を受ける
– 「思い込み」と「勘違い」を防ぐことはできない –

DENSO

1.7 一人プロジェクト … 部分理解の最悪ケース

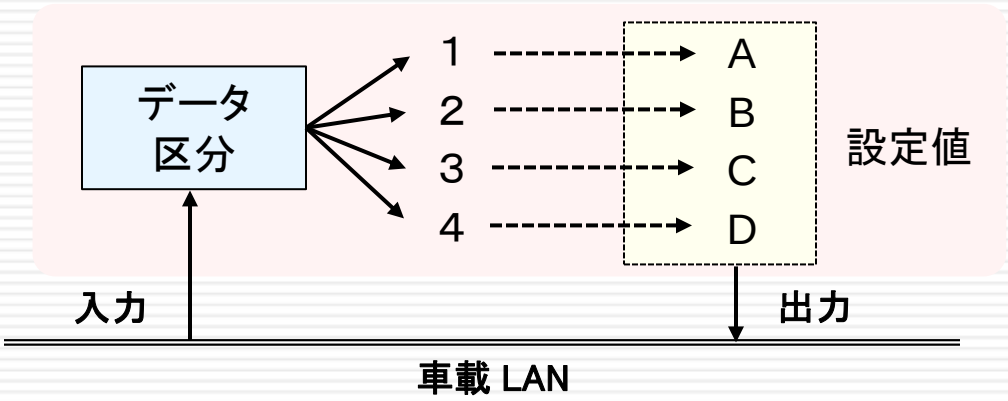
- 変更規模によっては担当者は一人
  - 意図的に担当者を一人にする場合もある
  - 経済的理由から一人プロジェクトは必ずしも否定できない
- 一人プロジェクトの問題
  - 「思い込み」と「勘違い」でソースコードを変更してしまう
  - レビューや要件管理などが省かれやすい
  - 担当者が孤立し、失敗した時のダメージを受けやすい

一人プロジェクトに対しては組織でカバーするのが原則

DENSO

1.8 変更方法は1つに決まる？

- 変更の実現方法は1つとは限らない
  - ［例］ 制御方法を一部変更する
    - 入力のデータ区分が“2”で 条件「X」の時、設定値をBからCに変更する

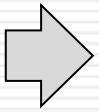


  - 見かけ上、同じ結果を得る変更方法は複数ある

- テストではどの実現方法もOKとなる
  - 選択した方法によっては、後に 「潜在バグ」 となって現れる

1.9 増えるトラブル ・・・

- 派生開発の不具合が招いた問題
  - 羽田の航空管制システムの停止（2003年/2008年）
  - 東京証券取引所でのトラブル
  - 携帯電話の通話トラブル
  - 走行中の自動車のエンジン停止（2005年）
  - 電子制御システムの問題（2010-2011年）　←　不具合なし



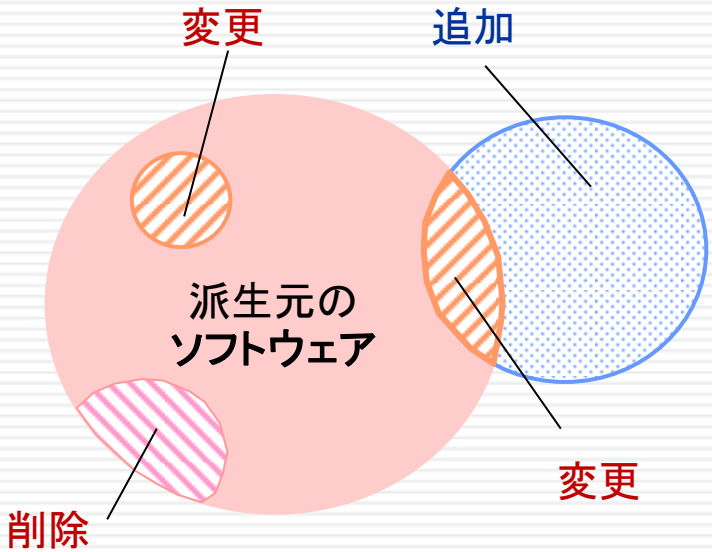
社会の混乱
企業の損益

- 「機能追加」 と 「移植」 のトラブルが多い
  - トラブルは、新しい機能を組み入れたところで起こるケースが多い
    - 機能を追加するために既存のソースコードを変更した箇所
    - 他のシステムからソースコードを切り出して、「移植」（流用）した箇所

既存のソースコードの「変更箇所」が記述されていない

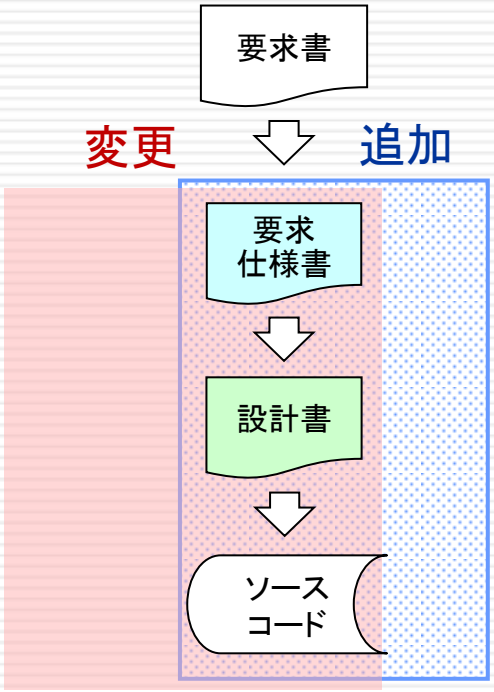
1.10 派生開発の難しさ（1）

- 既存のソフトウェアの変更
  - 関係する機能の特定　　：仕様上、影響を受ける機能の抽出
  - ソースコード変更の影響　：ソースコード修正による影響範囲の特定
- 技術者の問題
  - 技術力：ソースコードの読解力
  - 経験　：ドメイン知識（機能理解）
- ソースコード
  - 劣化した派生元のソースコード
    - 保守性を無視した開発



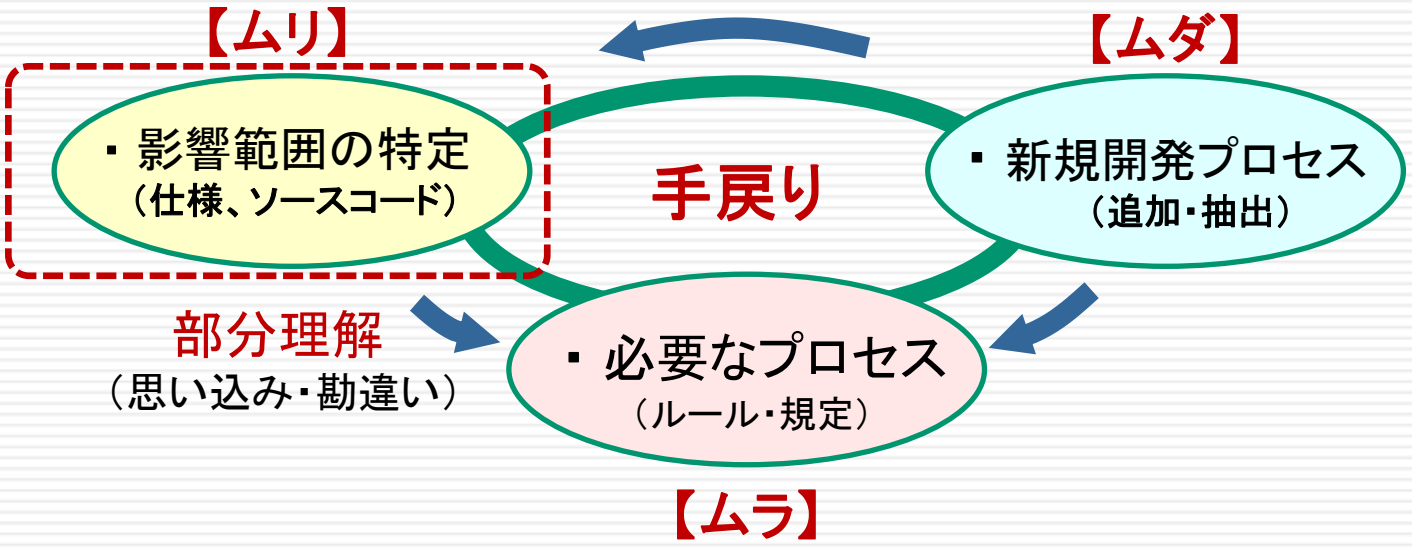
1.11 派生開発の難しさ（2）

- プロセス
  - 新規開発のプロセスは、要求の異なる「変更」と「追加」の両方に対応できない
    - 変更 ： 既存のソフトウェアの変更
    - 追加 ： 新しい機能の開発
  - 差分情報が整理されていない　[変更]
    - 変更点の追加と抽出を繰り返す
  - 見つけ次第ソースコードを直す［変更］
- 環境
  - 設計資料（ドキュメント）の不備
    - 理解の助けにならない



1.12 混乱する派生開発

- 混乱のメカニズム ： ムリ・ムダ・ムラ
  - 品質を確保する技術、ドメイン知識を持たない状態で
    納期やコストの削減が求められている → 技術者の疲弊



開発プロセス【ムダ】と影響範囲の特定【ムリ】が問題

1.13 派生開発に対するソリューション

- 派生開発の混乱要因
  - 「開発プロセス」と「影響範囲の特定」
- 「現状」と「改善ポイント」

	現 状	改善ポイント
開発プロセス	・**新規開発**のプロセスを適用 （新規開発崩し）	・**派生開発（変更・追加）**に 対応したプロセス
影響範囲の特定	・**部分理解**での作業 （思い込み、勘違い）	・思い込み、勘違いを 成果物とレビューでカバー

XDDP は派生開発の問題に対する有効なソリューション

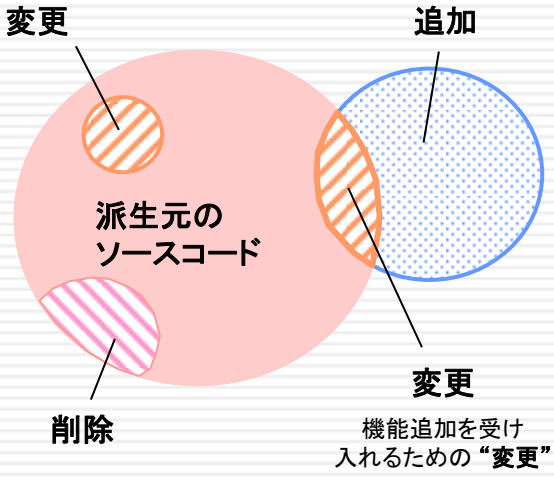
DENSO

- XDDPとは
- 従来手法との比較
- XDDPのプロセス
  - 追加のプロセス
  - 変更のプロセス
    - 変更要求仕様書
    - TM （トレーサビリティ・マトリクス）
    - 変更設計書
- XDDPの効果

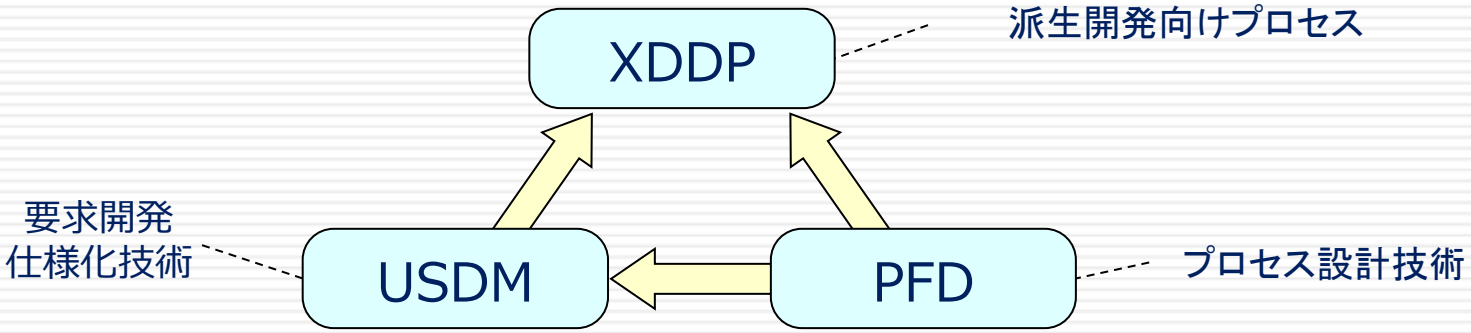
2.1 XDDP とは？

- XDDP ： eXtreme Derivative Development Process
  - 派生開発の要求に合った開発プロセス
  - 清水吉男氏（システムクリエイツ）が提案
- 合理的なプロセス
  - ２つの独立したプロセス
    - 「変更」と「追加」のプロセス
  - 品質と生産性を追求したプロセス
    - 差分情報に基づいた開発
    - ムダの徹底排除 ： Just in Time （TPS）
      - ✓ 「必要なものを」「必要なとき」「必要なだけ」
  - 部分理解を成果物とレビューでカバー
    - 担当者の「思い込み」と「勘違い」は前提

「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（2007）
清水吉男氏著

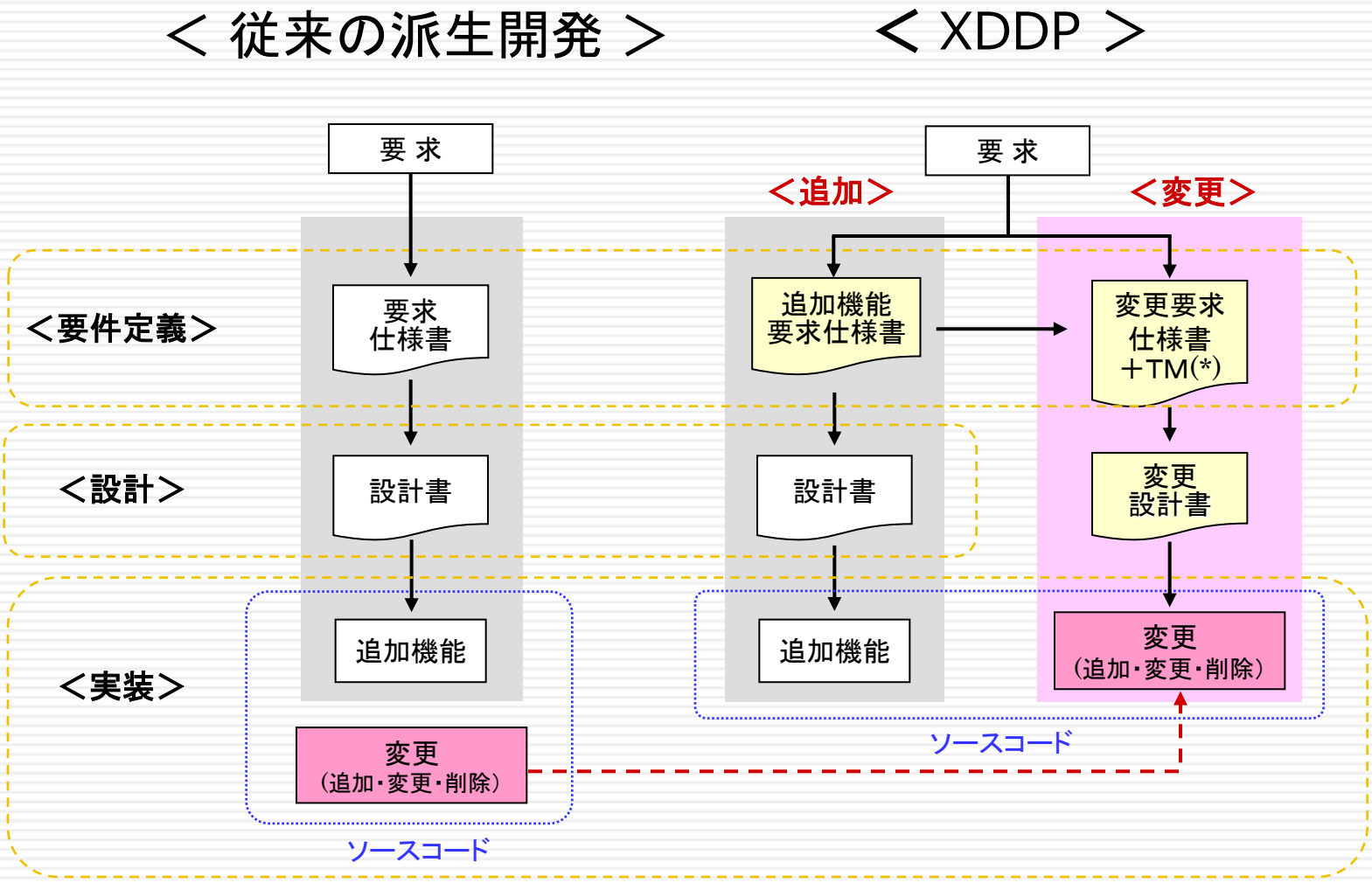


- XDDP トライアングル



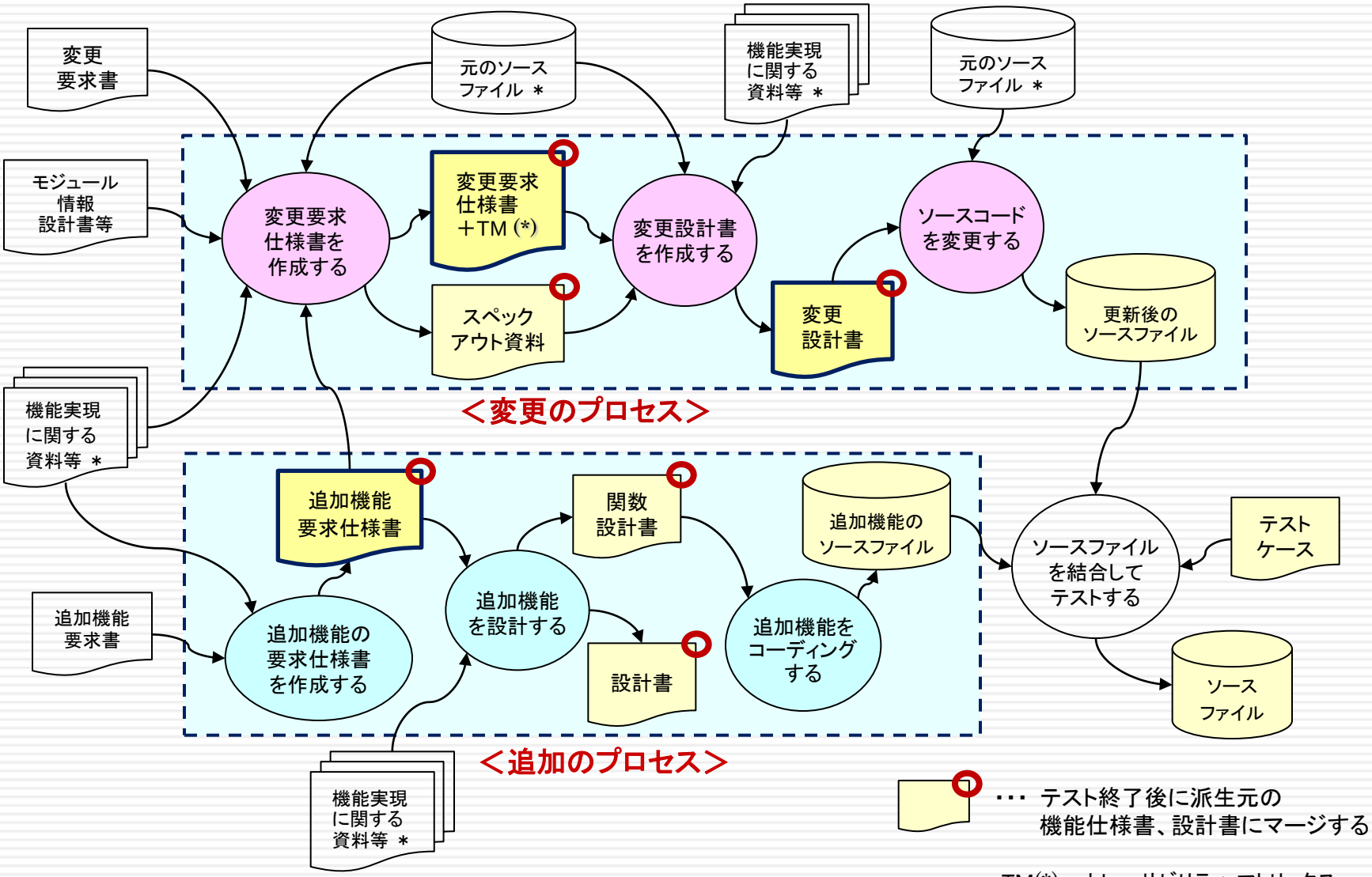
- USDM （Universal Specification Describing Manner）
  - 要求 を適切に表現し、必要な仕様を引き出す表記法
  - 要求と仕様を階層関係の中で捉え、仕様が漏れにくい構成
- PFD （Process Flow Diagram）
  - 合理的な “成果物とプロセスの連鎖” を設計する技術
  - 開発途中の変化に対して、「別案」のプロセスを考え出すツール

2.3 従来手法との比較



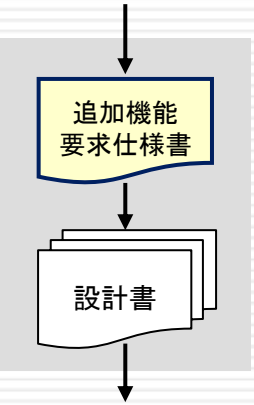
TM(*) ：トレーサビリティ・マトリックス

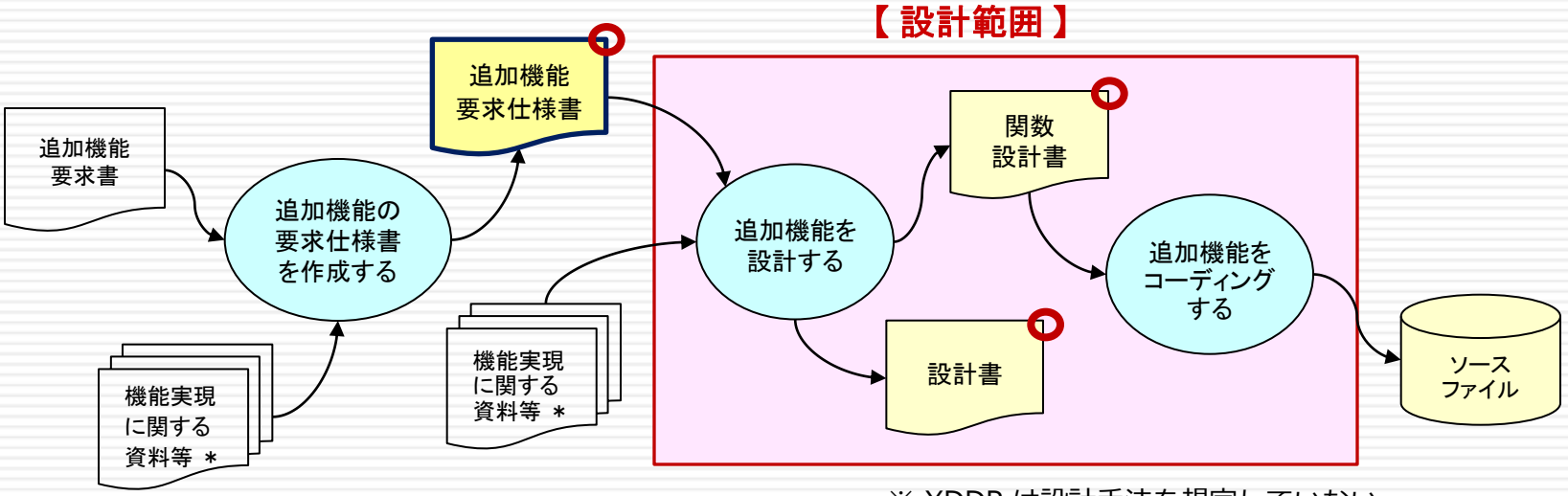
2.4 XDDP のプロセス



2.5 追加のプロセス

- 追加プロセスの特徴
  - 一般の新規開発のプロセスに準拠したプロセス
    - ✓ 採用する設計手法の違いによりプロセスは変化する
  - 通常は「アーキテクチャ設計」のプロセスが存在しない
    - ✓ 但し、既存のアーキテクチャの制約は受ける
  - 追加機能要求仕様書は新規開発の要求仕様書と基本的に同じ



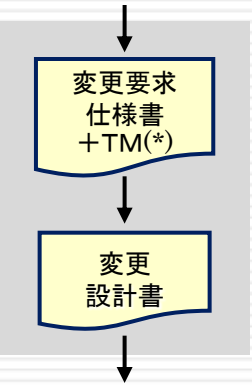


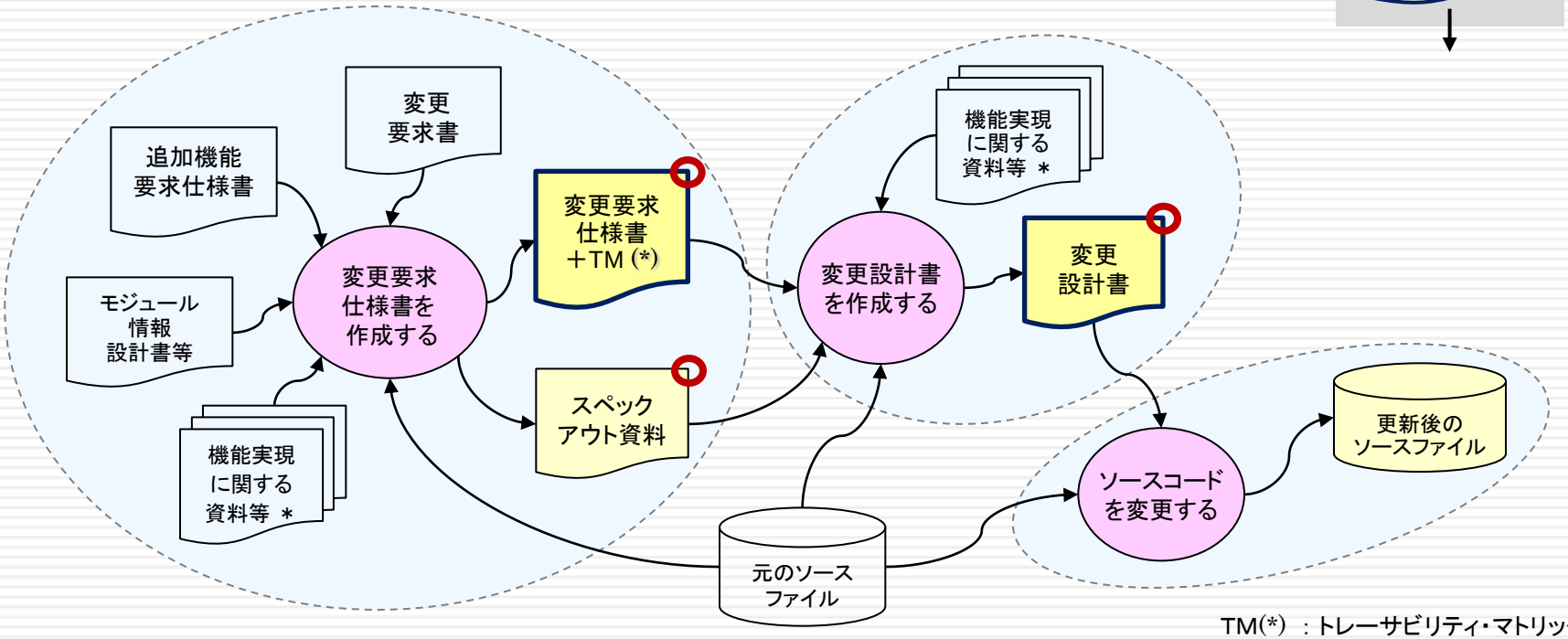
※ XDDP は設計手法を規定していない

2.6 変更のプロセス

■ 変更プロセスの特徴

- 派生元のソースコードの変更点に着目したプロセス
  - 変更箇所、変更方法を 「3点セット」の成果物によって記述し、全て揃った後に一斉にソースコードを変更する
    - ✓ 3点セット ： 変更要求仕様書、TM、変更設計書





TM(*) ：トレーサビリティ・マトリックス

- 変更に着目した成果物
  - ソースコードの変更前に、３つの成果物で全ての変更内容をそれぞれの視点で抽出し、レビューする

成果物	カバー範囲	内容	レビュー(**)	
変更要求仕様書	What Why	何をなぜ変更するか どのような振る舞いをするか	○	○
TM (*)	Where	変更仕様がどこにあるか	○	
変更設計書	How	変更仕様をどのように修正するか	○	○

TM (*)　：トレーサビリティ・マトリックス
レビュー(**)　：変更要求仕様書 と TM は一緒にレビューをしてもよい

2.8 変更要求仕様書 –USDM–

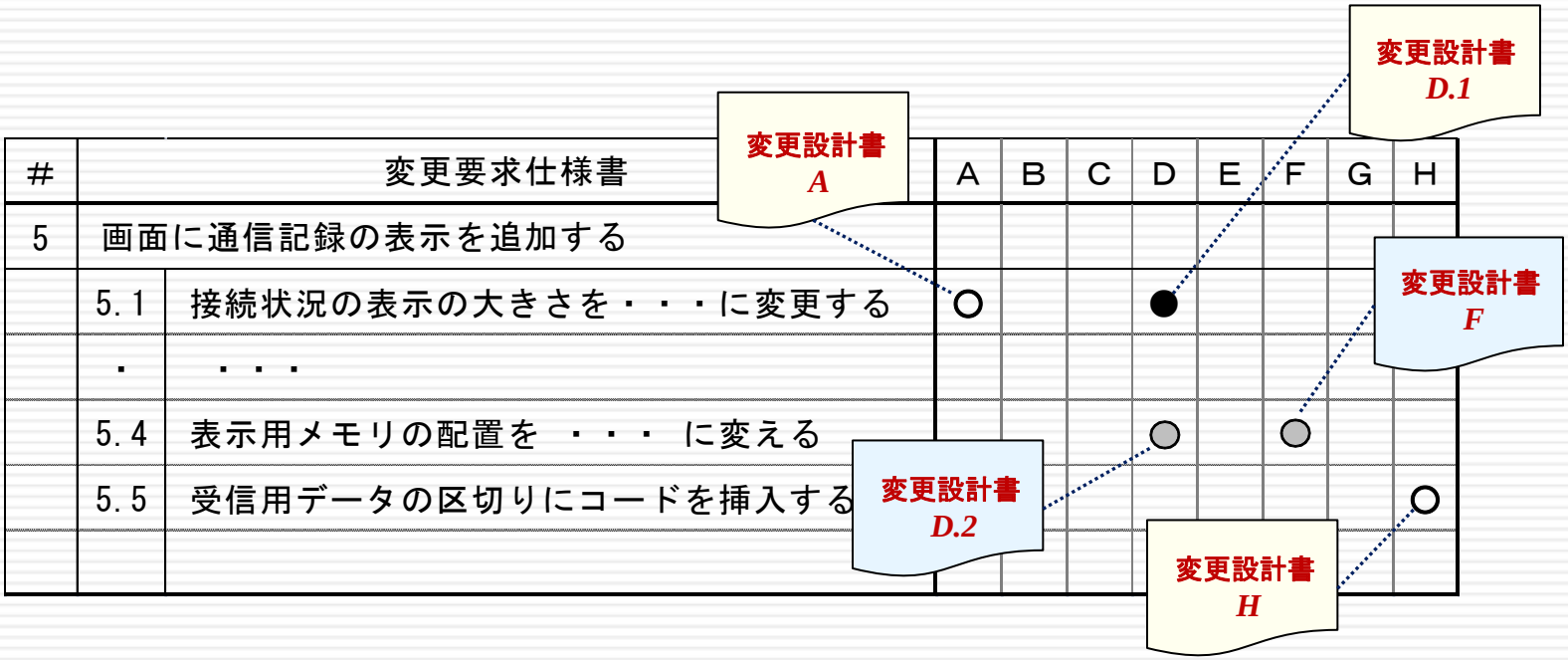
- 要求と仕様を階層構造で表現
  - 要求
    - 仕様
    - 仕様
- 変更の理由を記述
  - 適切な変更仕様を引き出す
- 表現の工夫 （変更の表現）
  - Before ／ After
    - 「 ～ を ○○ に変更する 」
    - 「 ～ を 削除する 」
    - 「 ～ を △△ に追加する 」

要求	Req.1		
	理由		
	説明		
	<Group 1>		
	要求	Req.1-1	
		理由	
		説明	
	仕様	<グループA>	
		Req.1-1-1	
		Req.1-1-2	
		<グループB>	
		Req.1-1-3	
		Req.1-1-4	
	<Group 2>		
	要求	Req.1-2	
		理由	
		説明	
	仕様	<グループC>	
		Req.1-2-1	
		Req.1-2-2	
		<グループD>	
		Req.1-2-3	
		Req.1-2-4	

USDM：Universal Specification Describing Manner

DENSO

- TM 上で変更仕様と変更設計書を対応させる
  - 変更仕様に該当する箇所を TM 上に表す
  - 変更箇所の関連性から影響範囲の気付きが得られる



#	変更要求仕様書	A	B	C	D	E	F	G	H
5	画面に通信記録の表示を追加する								
	5.1 接続状況の表示の大きさを・・・に変更する	○			●				
	・ ・・・								
	5.4 表示用メモリの配置を ・・・ に変える				◎		◎		
	5.5 受信用データの区切りにコードを挿入する								○

2.10 変更設計書

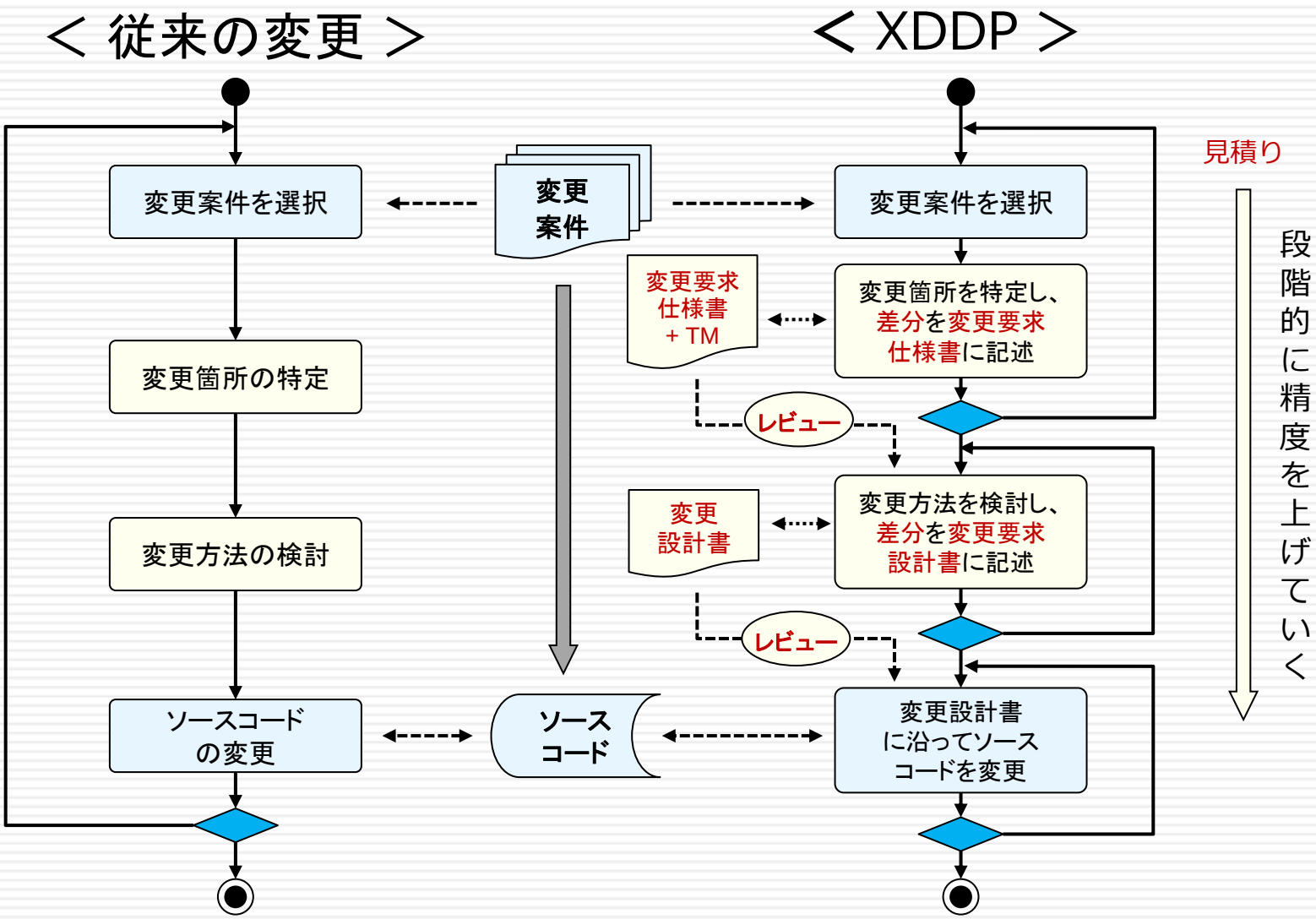
- 具体的なソースコードの変更方法を記述する
  - 「関数名」単位で、具体的な変更方法を文章で記述する
    - 「差分」（before/after）の記述に徹する　※ 変更設計書 ≠ 関数仕様書
  - 変更に伴うテスト内容（単体テスト）も記述する

- 変更設計書の構成

（1） ヘッダー部	・・・	TM の交点の情報、修正方針など
（2） 構造の変更	・・・	データ構造、処理構造（関数呼び出し）の変更
（3） 関数外の変更	・・・	関数外の「定義」の変更
（4） 関数の変更	・・・	関数内の変更箇所
（5） 確認内容	・・・	変更を確認する単体テスト項目

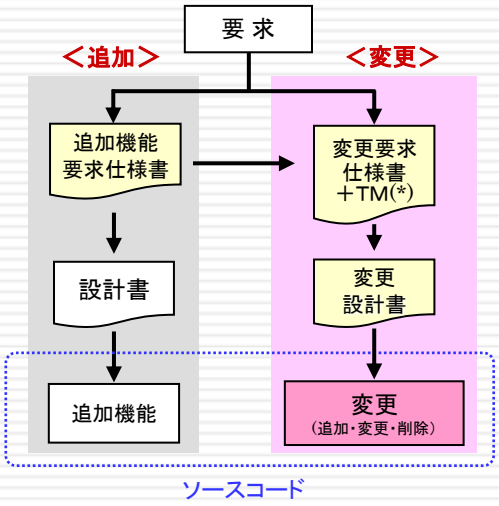
※ 『「派生開発」を成功させるプロセス改善の技術と極意 』
付録 変更設計書テンプレート（P406-410） 参照

2.11 従来の変更作業との違い



2.12 XDDPによる派生開発

- 混乱要因の解消
  - 開発プロセス
    - 変更／追加 の独立したプロセス
  - 影響範囲の特定
    - ① 変更要求仕様書
    - ② TM（トレーサビリティ・マトリクス）
    - ③ 変更設計書



① 変更要求仕様書

要求	Req.1		
	理由		
	説明		
	<Group 1>		
	要求	Req.1-1	
		理由	
		説明	
	仕様	<グループ A>	
		Req.1-1-1	
		Req.1-1-2	
		<グループ B>	
		Req.1-1-3	
		Req.1-1-4	
	<Group 2>		
	要求	Req.1-2	
		理由	
		説明	
	仕様	<グループ C>	
		Req.1-2-1	
		Req.1-2-2	
		<グループ D>	
		Req.1-2-3	
		Req.1-2-4	

② TM（トレーサビリティ・マトリクス）

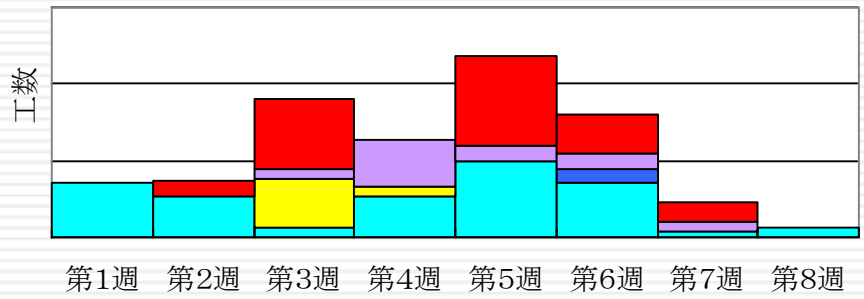
#	変更要求・仕様	A	B	C	D	E	F	G	H
5	画面に通信記録の表示を追加する		●						
	5.1 接続状況の表示の大きさを・・に変更する							●	
	. ・・・					●			
	5.4 表示用メモリの配置を・・に変える								
	5.5 受信時データの区切りにコードを挿入する								

③ 変更設計書

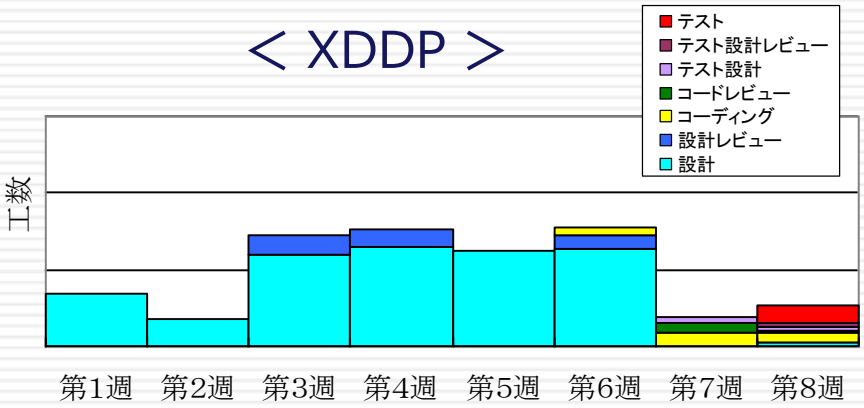
2.13 XDDP の効果

- 工数分布の比較

＜従来のプロセス＞



＜ XDDP ＞



- テストで不具合を摘出
- 設計、コーディング、テストを繰り返すことで品質を確保

⇩

テストにより品質確保

- 上流に重点を置いた開発
- コーディングは、ほぼ1回で完了

⇩

適切な仕様化により後戻りなし

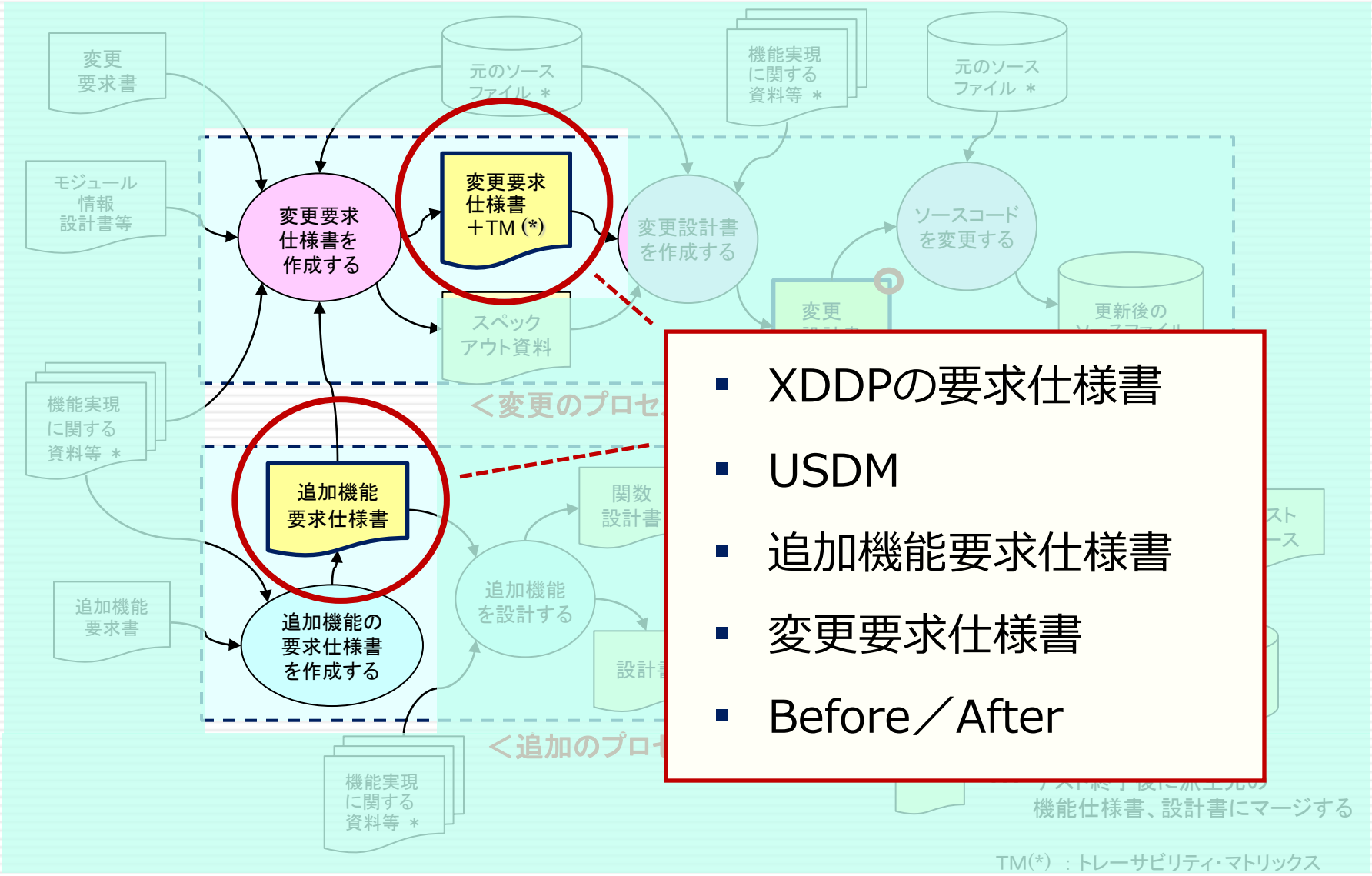
XDDP は 派生開発の有効なソリューションである

3．XDDPの勘所！

- 要求仕様書の考え方
- スペックアウト
- TM
- ソースコードの変更
- 公式文書の作成

DENSO

3.1 要求仕様書の考え方



- XDDPの要求仕様書
- USDM
- 追加機能要求仕様書
- 変更要求仕様書
- Before／After

DENSO

3.1.1 XDDP における要求仕様書

- 「要求仕様書」 とは

> 「要求」（ 実現して欲しいこと：Requirements ）について、
> 作る関係者が 認識を特定（Specify）できている文書

  - 顧客の要求を実現することを目的にして作られた文章（cf. 機能仕様書）
    - 関係者だけが理解できればよい
    - 担当者（新人、ベテラン）によって、書き方や構成を変えてもよい

- 「要求」と「仕様」
  - 「要求」： 機能、性能、制約条件を抽象的に表現したもの
    - 一連の動き（入力-変換-出力）を持つ振る舞い
  - 「仕様」 ： 要求に含まれる具体的な処理や振る舞いを表現したもの
    - 実現可能性 ： 設計の様子がイメージできる
    - 検証可能性 ： 仕様の実現を検証する方法がある

- 要求と仕様を階層構造で表現
  - 要求
    - 仕様
    - 仕様
- 要求 （理由）
  - 振る舞いの適切な「範囲」を示し、必要な仕様を導き出す
- 仕様
  - 要求が表現している動詞に着目し、仕様に展開していく

要求	Req.1		
	理由		
	説明		
	<Group 1>		
	要求	Req.1-1	
		理由	
		説明	
	仕様	<グループA>	
		Req.1-1-1	
		Req.1-1-2	
		<グループB>	
		Req.1-1-3	
		Req.1-1-4	
	<Group 2>		
	要求	Req.1-2	
		理由	
		説明	
	仕様	<グループC>	
		Req.1-2-1	
		Req.1-2-2	
		<グループD>	
		Req.1-2-3	
		Req.1-2-4	

USDM は 表現を重視した仕様記述法

DENSO

3.1.3［事例］：キッチンタイマー

1.2 カウントダウン機能	要求	CNT01	タイマー値設定状態でStart/Stopボタンを押すと、タイマーのカウントダウンを開始できる。	
		理由	任意のタイミングでカウントダウンを開始したい。	
		説明		
	□□□	CNT01.01	タイマー値設定状態からStart/.Stopボタンが押下されたら、タイマー値がゼロかそれ以外かを調べる。	
	□□□	CNT01.02	タイマー値がゼロのとき、何もせず、タイマー値設定状態を維持する。	
	□□□	CNT01.03	タイマー値がゼロ以外のとき、カウントダウン状態へ移行して、1秒毎のタイマー割り込みを開始する。	
	要求	CNT02	カウントダウン状態では、表示されているタイマー値をカウントダウンする。	
		理由	ユーザに残りの時間を読み取れるようにする。	
	要求	CNT03	カウントダウン中に、Start./Stopボタンを押すことで、タイマーを停止し、タイマー値を保持したまま、タイマー値設定状態に戻すことが出来る。	
		理由	タイマーの進行を任意のタイミングで停止させたい。	

DENSO

3.1.4 追加機能要求仕様書

- 「XDDP」における追加機能を扱う要求仕様書

> 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

- 新規開発時の要求仕様書と基本的に同じ構成
- “関係者” は、特定されている （cf. 機能仕様書）
- 実現方法は 「設計プロセス」 で選択し、評価する
- 「作り方の品質要求」 も含まれる

DENSO

3.1.5 変更要求仕様書

- 「XDDP」における変更(*)を扱う要求仕様書

変更 (*) ：対象とするソースコードに対する変更

今回の派生開発で変更したいこと（Changre 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について特定した（Specify）ことがまとめられた文書

- 変更の実現方法まで特定（Specify）する
- 変更仕様は、ソースコード（関数仕様）のレベルで記述する
  - 新規開発/機能追加 ： 設計プロセスで実現方法をレビュー
  - 変更 ： 明確な「設計プロセス」が存在しない = レビューの機会がない
- 既に設計されたものを変更するだけ
- 変更する関数名（場所）は TM で検討する

3.1.6 変更の表現 – before / after –

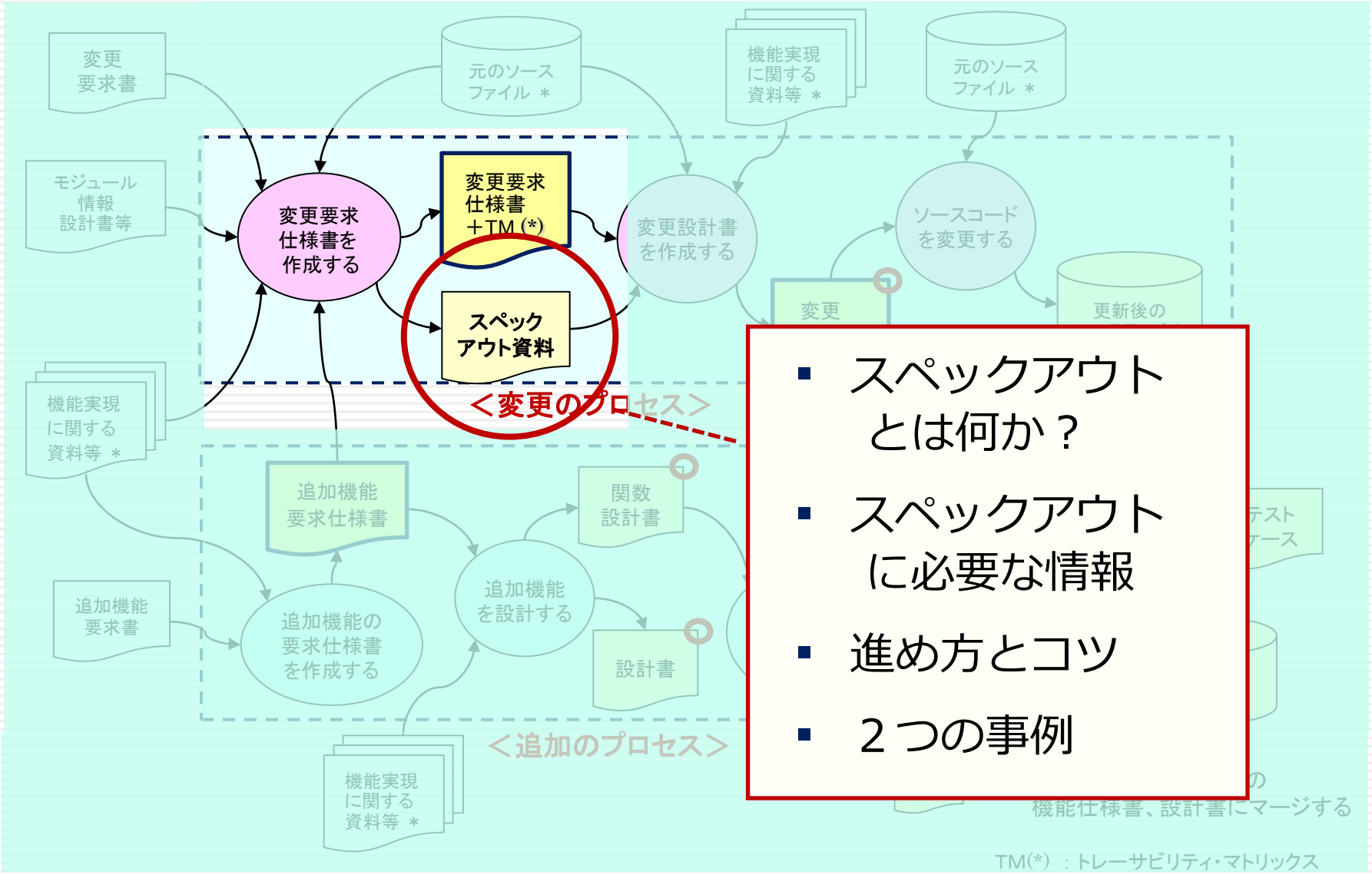
- 現行の製品やシステムに対する変更を表現する
  - 動作しているシステム（製品）の仕様の 変更、追加、削除
  - 今回のプロジェクトで機能追加を受け入れるための変更
- 「変更要求」 と 「変更仕様」 の階層構造で表現する
  - 変更要求 ： 変更の意図と範囲、理由（背景）
  - 変更仕様 ： 具体的な変更の内容、変更箇所
- 変更要求、変更仕様は 「before」「after」で表現する
  - 「追加」「削除」 は、「before」「after」 を含んでいる

変更要求	変更	金額の表示欄の選択が商品区分に限定していたものを、表示欄との対応付けを設定できるように変更する
	追加	商品一覧と在庫確認リストの上に写真の表示を追加する
	削除	商品選択前の確認画面の表示機能を削除する

3.1.7 Before / After 表現の利点

- 変更箇所をイメージできる
  - 「before」（現状）を表現することで、ソースコード、変更の該当箇所、影響範囲を探すのに非常に有効
    - 「after」 だけでは変更に関するイメージがわかない
- 変更規模の見積りのヒントを得る
  - 「before」と「after」から変更の様子を把握できる
    - 変更仕様の数 / 規模 / 難易度
    - ソースコードの変更行数
    - 変更箇所の影響範囲
  - サイズ見積りと組み合わせることで、担当者との変更に対する認識の違いがわかる

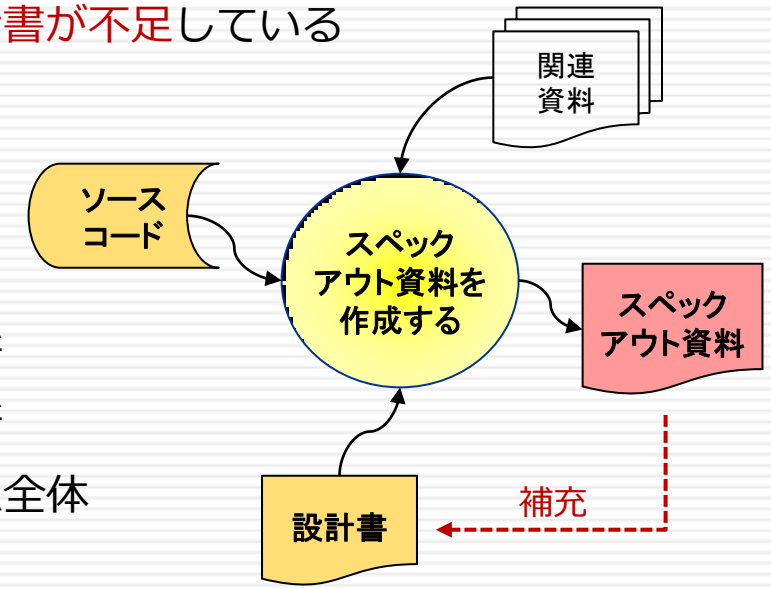
3.2 スペックアウト



- スペックアウトとは何か？
- スペックアウトに必要な情報
- 進め方とコツ
- ２つの事例

3.2.1 スペックアウトとは何か？

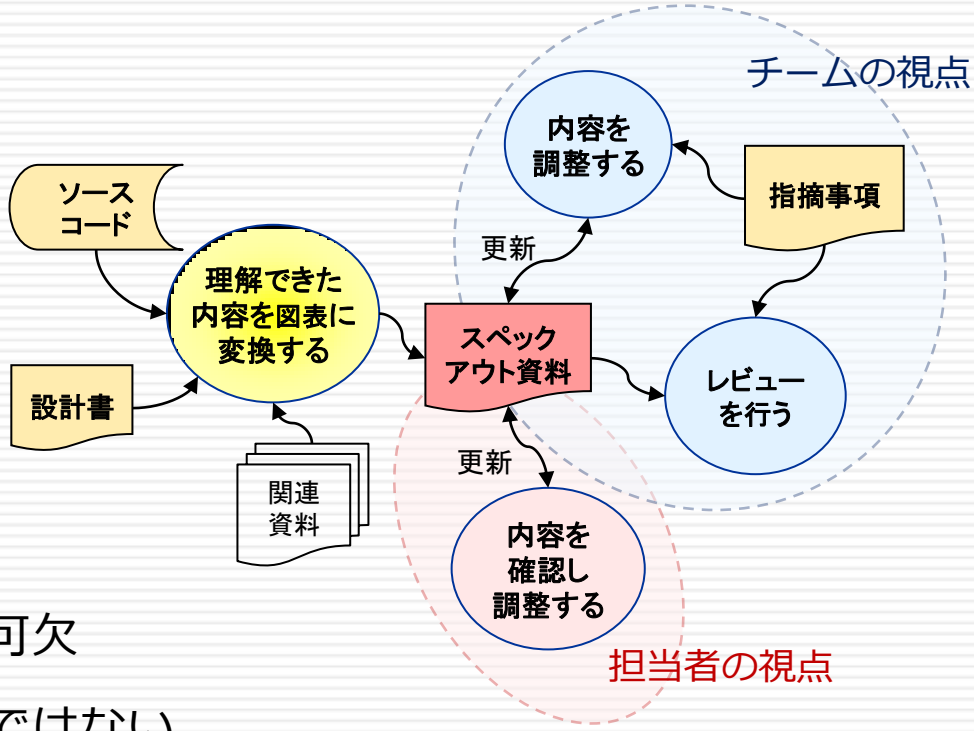
- スペックアウトとは？
  - ソースコードから設計書の一部を生成する行為
    - 派生開発では必要とする派生元の設計書が不足している
- スッペックアウトの目的
  - 構造や設計の意図の理解
    - 適切な変更方法の決定　→　変更箇所
    - リファクタリング　→　特定箇所
    - アーキテクチャの改修　→　システム全体
- 成果物
  - 設計書を構成する図、表を使って表現する
    - 構造図、フロチャート、DFD、PADチャート など



<スペックアウト>

3.2.2 スペックアウトの本質

- 思い込み、勘違いの解消
  - 理解できた内容を適切な形（図・表）に変換して表現する
    - 形を変えることで、担当者が理解できているか判断できる
    - 他人も理解の適切さを判断（レビュー）できる



<思い込み・勘違いの解消>

- 設計技術が必要
  - 設計技術と設計経験は必要不可欠
  - 設計思想を理解するのは容易ではない
  - システムの知識、ドメインの知見も必要

3.2.3 スペックアウトに必要な情報

- 派生開発に必要な情報
  - 構造の理由と背景
    - データ構造　・・・　「なぜ、そのようなデータ構造にしたのか？」
    - 処理構造　・・・　「なぜ、この処理をこの関数で扱っているのか？」
    - 制御構造　・・・　「なぜ、この状態遷移で仕様を満たすことができるか？」
- スペックアウトの必要性
  - 新規開発：
    - 派生開発に必要で効果的な設計資料を考慮せずに開発した
    - “どう作るか” を中心に設計資料を作成してきた
      - ✓ cf：仕様はあるが要求がない状態　（機械設計：図面しかない状態）
  - スペックアウト ：
    - 新規開発における設計プロセスの不足を補う行為
      （派生開発の機能追加も含む）

DENSO

3.2.4 スペックアウトの進め方とコツ

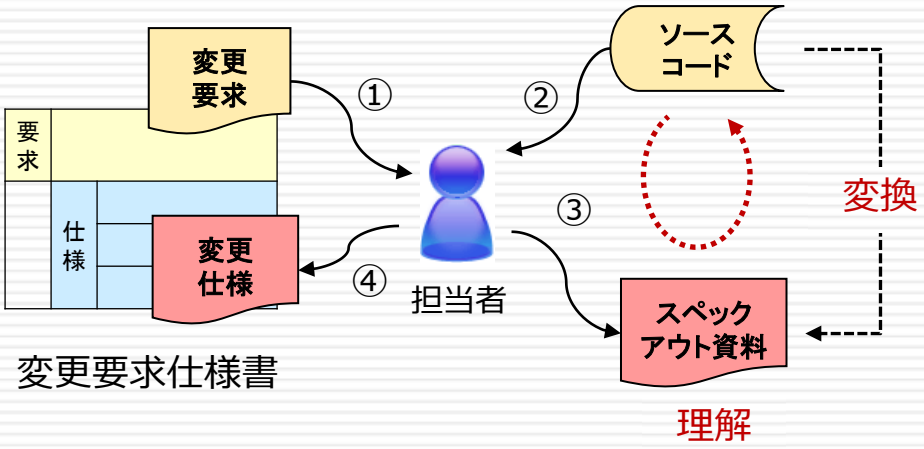
- スペックアウトの進め方

① 変更要求の理解

② ソースコードの変換

③ スペックアウト資料の記入

④ 変更仕様の記入



<スペックアウトの手順>

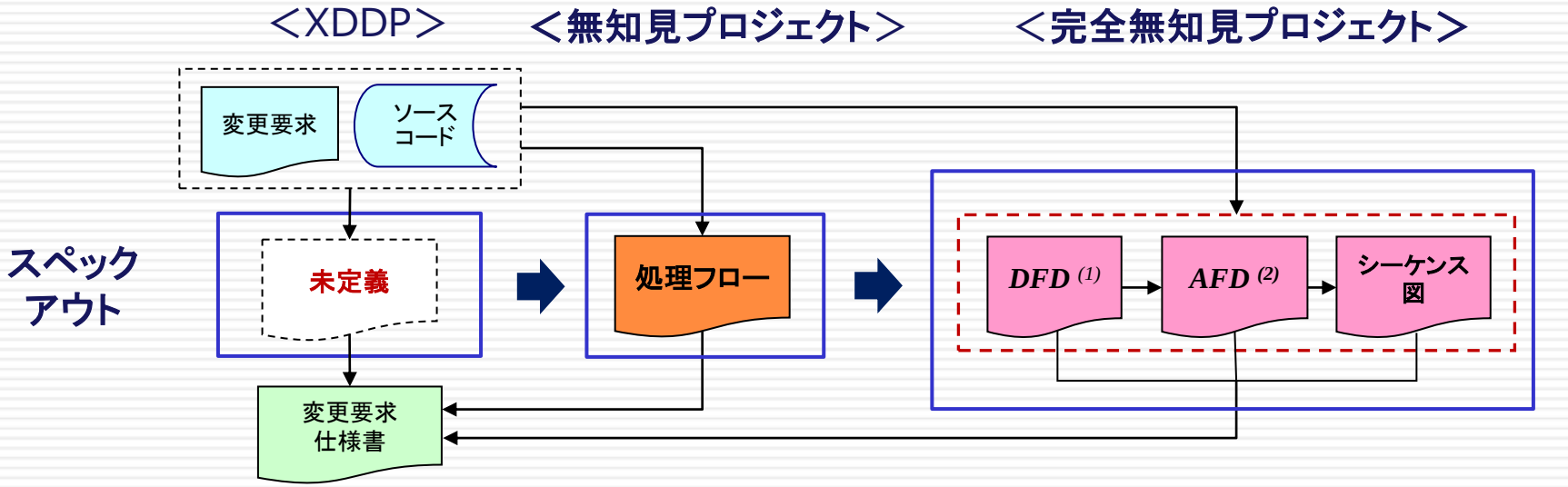
- スペックアウトのコツ
  - ○：資料に書きながら理解する　×：理解したことを資料に書く
    - ソースコードを読む過程で、スペックアウト資料を作成し、スペック資料を通してソースコード（設計思想）を理解する
  - ソースコードのわかった内容の記述には時間はかからない
    - スペックアウト資料により、理解した部分を振り返る時間は短縮する

3.2.5 スペックアウトの事例

- プロジェクトへの対応
  - 無知見プロジェクト
    - 派生元のソースコードの知見を有した担当者が不在のプロジェクト
  - 完全無知見プロジェクト
    - 無知見プロジェクト ＋ 開発ドメインの知識が不十分

⇒ 設計資料がない場合が多い

- 変更仕様の抽出
  - 条件にあった設計手法を選択し、スペックアウトに適用する

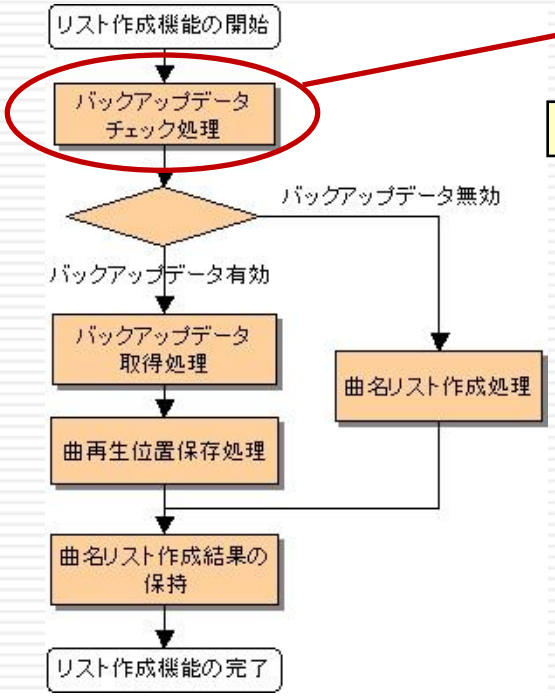


DFD(1) : Data Flow Diagram　　*AFD(2) : Architecture Flow Diagram*

3.2.6 事例1：処理フロー

1. 処理フローを作成

<処理フロー>



<変更要求仕様書>

要求	オーディオ制御	電圧低下状態からの復帰時、曲の先頭から再生する。本仕様を、電圧低下検出直前の再生位置から、再生するように変更する。
	<バックアップデータチェック処理の変更>	
	<曲名リスト作成条件の変更>	
	<バックアップデータ取得処理の変更>	

2. 処理ごとにグループを抽出

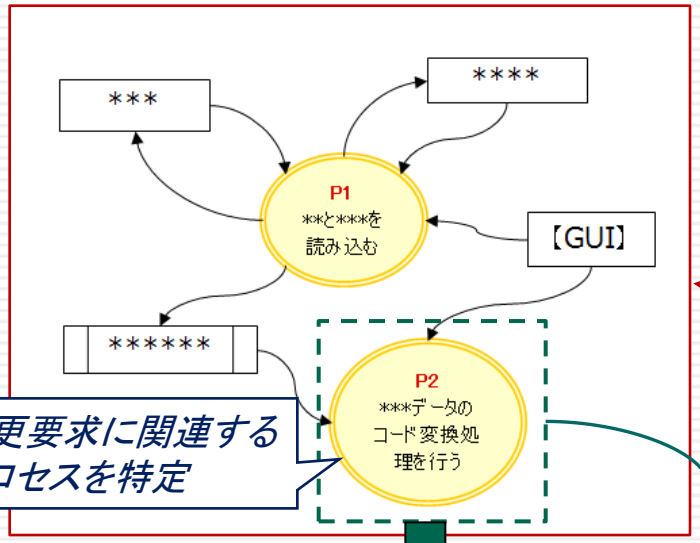
要求	オーディオ制御	電圧低下状態からの復帰時、曲の先頭から再生する。本仕様を、電圧低下検出直前の再生位置から、再生するように変更する。
	<バックアップデータチェック処理の変更>	
	<曲名リスト作成条件の変更>	
	要求 2.1	曲名リストを作成するタイミングを、バックアップデータチェックの結果と、曲名リストの作成結果を合わせて判断する。
	要求仕様2.1.1	バックアップデータチェック結果が「正常終了」以外の場合、曲
	要求仕様2.1.2	曲名リストが「作成完了以外」の場合、曲名リストを作成する。
	<バックアップデータ取得処理の変更>	
	3.1	変更なし

3. グループで検討範囲を限定し
要求と仕様の漏れを防ぐ

処理フローから変更仕様を抽出する

DENSO

3.2.7 事例2：DFD+AFD+シーケンス図（1）



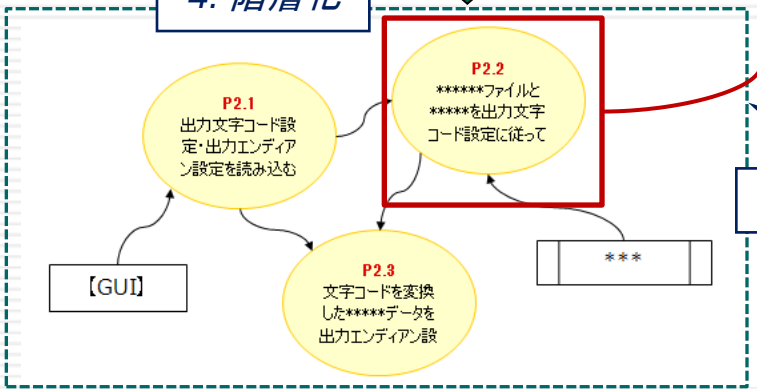
DFD（*Data Flow Diagram*）

1. 変更要求に関連する機能の DFD を作成

2. 変更要求に関連するプロセスを特定

3. 該当プロセスをグループとして抽出

4. 階層化



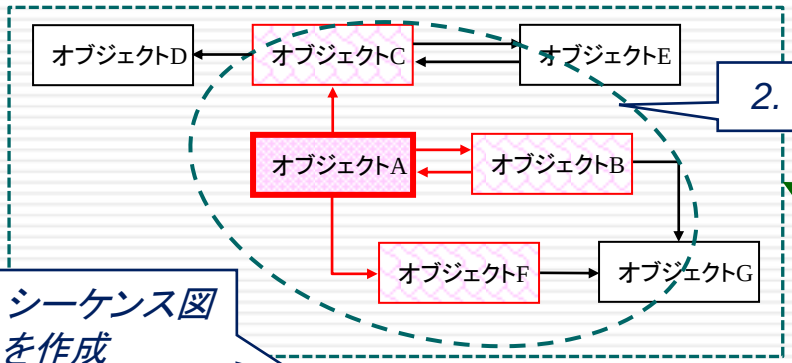
5. 下位の変更要求として抽出

<変更要求仕様書>

要求	RC01	****ファイルから* 追加して"UTF-16 **上位要求** SO_81","S-JIS","UTF-8"に できるようにする。
理由		***システムにおける標準文字コードがUTF-16である。そのため、利用 されるデータは予めUTF-16で用意されている必要がある
		<読み込んだファイルの文字コード変換指定の変更>
要求	RC01.5.1	****処理の直 -16に変換す る処理を追加 **下位要求**
理由		-
		<文字コード変換>
□□□	RC01.5.1.1	****関数呼び出し前に関数内で使用する変数を 定義し初期化を行う
		C01.5.1.1後、*****データがUTF-16へ変換され 場合のデータサイズを取得する
■■■	RC01.5.1.3	*****データをUTF-16に変換し、RC01.5.1.3で確 保した領域に格納する

DFDでソースコードの構造を把握し変更要求（下位）を抽出

3.2.8 事例2：DFD＋AFD+シーケンス図（2）



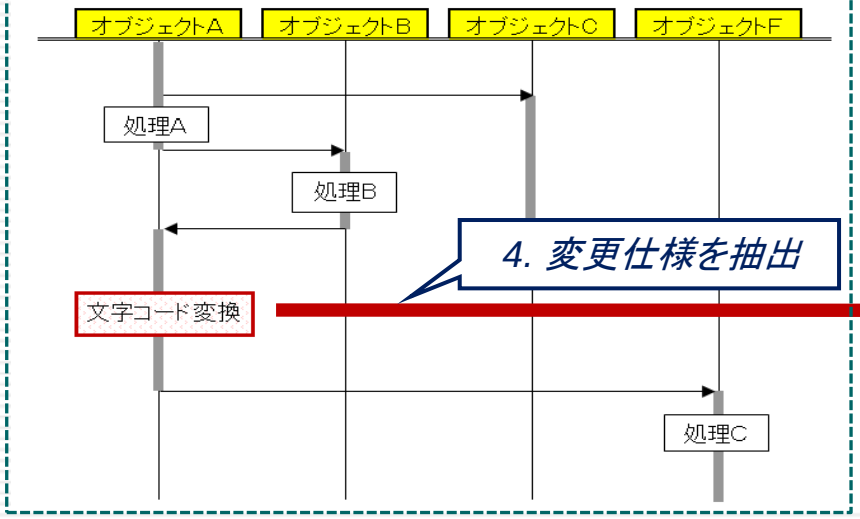
AFD（*Architecture Flow Diagram*）

2. 変更により影響を受けるオブジェクトを特定

1. 抽出した下位要求をAFD で記述する

3. シーケンス図を作成

シーケンス図



4. 変更仕様を抽出

＜変更要求仕様書＞

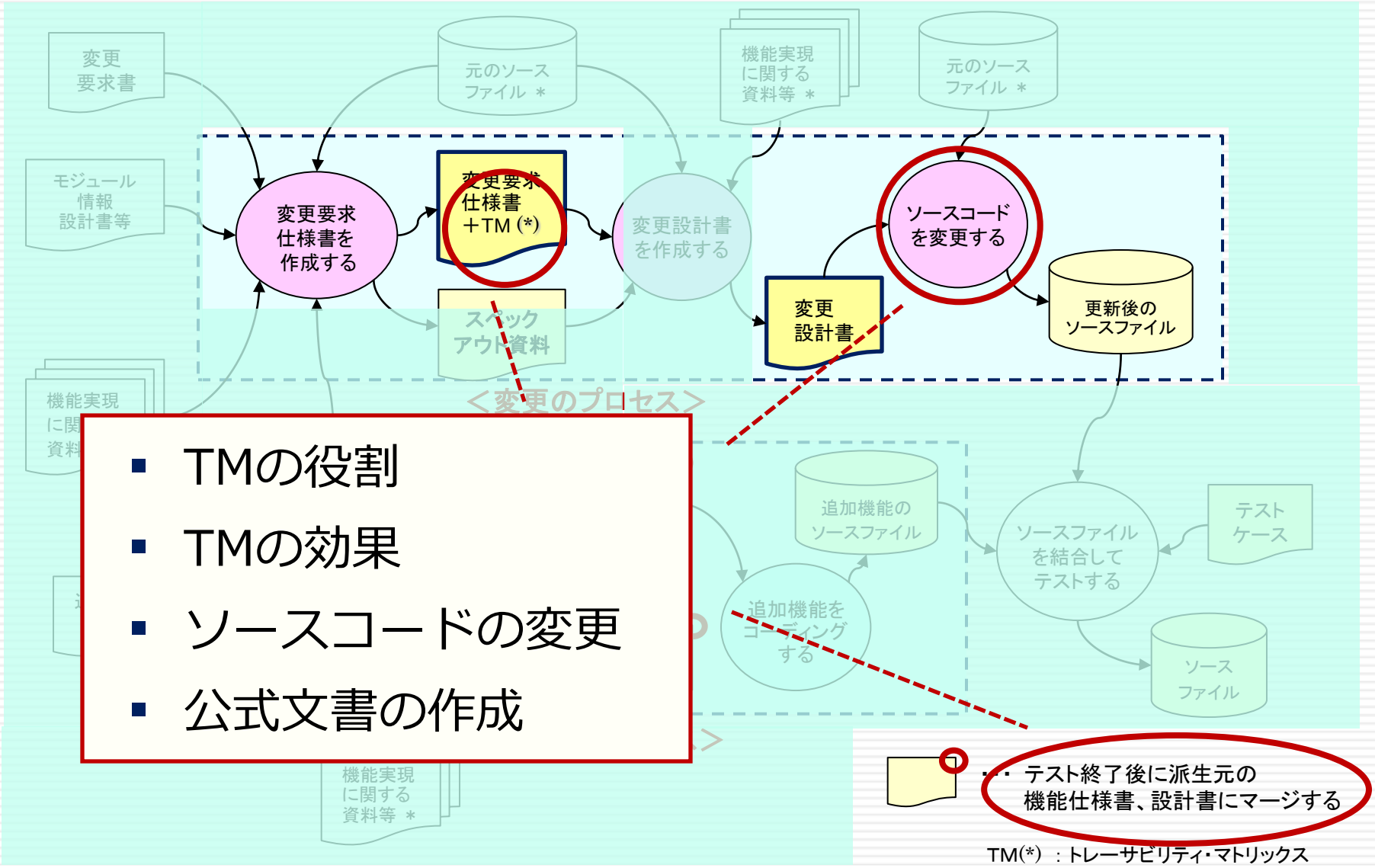
要求	RC01	****ファイルから*****を生成する際に,"ISO_81","S-JIS","UTF-8"に追加して"UTF-16"への文字コードを変換できるようにする。
	理由	***システムにおける標準文字コードがUTF-16である。そのため、利用されるデータは予めUTF-16で用意されている必要がある
	<***機能で利用するユーザー設定の指定の変更>	
	<外部ファイル読み込み処理の変更>	
	<読み込んだファイルの文字コード変換指定の変更>	
要求	RC01.5.1	*データをUTF-16に変換す
	理由	
	<文字コード変換>	
□□□	RC01.5.1.1	****関数呼び出し前に関数内で使用する変数を定義し初期化を行う
■■■	RC01.5.1.2	RC01.5.1.1後、*****データがUTF-16へ変換された場合のデータサイズを取得する
■■■	RC01.5.1.3	*****データをUTF-16に変換し、RC01.5.1.3で確保した領域に格納する

下位要求

AFD、シーケンス図 により変更仕様を抽出

DENSO

3.3 TM、ソースコードの変更、公式文書の作成



- TMの役割
- TMの効果
- ソースコードの変更
- 公式文書の作成

DENSO

3.3.1 TM：トレーサビリティ・マトリクス

- TM（トレーサビリティ・マトリクス）とは ？
  - 新規開発（機能追加）時に、個々の仕様がどのモジュールで実現しているかを表すマトリクス　　cf. インパクト・マトリクス
  - “列” のモジュール情報は、本来の トレーサビリティ・マトリクス と同じ情報をそのまま使用し、さらに必要に応じて “列“ 情報を追加する

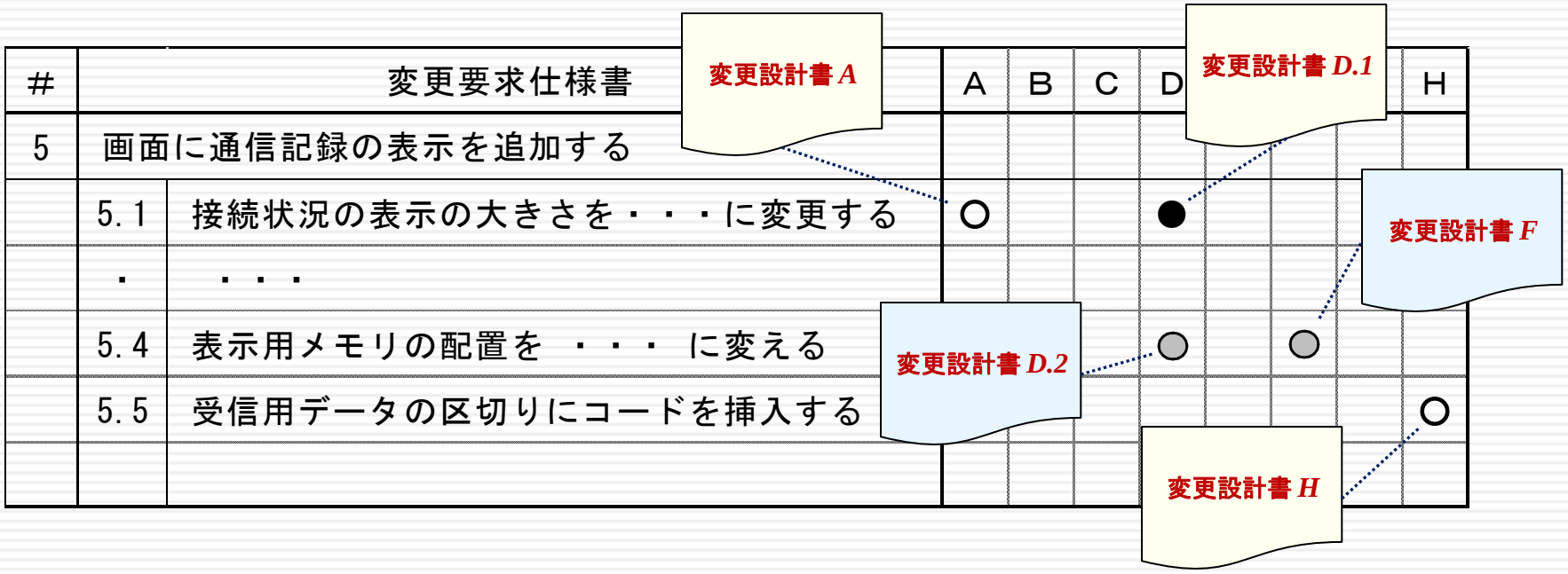
変更情報

変更対象のモジュール構造

#	変更要求仕様書		A	B	C	D	E	F	G	H
5	画面に通信記録の表示を追加する									
	5.1	接続状況の表示の大きさを・・・に変更する								
	・	・・・								
	5.4	表示用メモリの配置を ・・・ に変える								
	5.5	受信用データの区切りにコードを挿入する								

必要なら他の情報を加える

- 変更仕様と変更するモジュールをTM上で対応させる
  - 列情報は必要に応じて拡張する
    - リンクパラメータ、ビルドのパラメータ、機能仕様書、設計書などの文書
  - ソースコードの具体的な変更は、変更設計書で記述する



#	変更要求仕様書		A	B	C	D			H
5	画面に通信記録の表示を追加する								
	5.1	接続状況の表示の大きさを・・・に変更する	○			●			
	・	・・・							
	5.4	表示用メモリの配置を ・・・ に変える				○		○	
	5.5	受信用データの区切りにコードを挿入する							○

3.3.3 TM の効果

- 変更箇所の関連性から影響範囲の気付きが得られる
  - 関数やクラスの単位では具体的すぎて推測しにくい
- 変更要件に対する変更箇所、関係するモジュールが確認できる
- 複数の担当者が変更に関わる時の問題を未然に防ぐ
  - 変更要件の解釈の違いや修正方法の不整合等

モジュール毎の確認

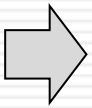
#	変更要求仕様書		A	B	C	D	E	F	G	H
5	画面に通信記録の表示を追加する									
	5.1	接続状況の表示の大きさを・・・に変更する	○			●				
	・	・・・								
	5.4	表示用メモリの配置を ・・・ に変える				◎		◎		
	5.5	受信用データの区切りにコードを挿入する								○

変更要件毎の確認

(注) 変更仕様に該当するモジュールの欄には、関数名を記述する

3.3.4 ソースコードの変更

- ソースコードは一気に変更する（変更）
  - 事前に変更設計書のレビューを終えて足並みを揃える
    - 追加機能は、変更要求仕様書の中で追加機能を受け入れる
      変更仕様が特定、検証されていれば、変更とは独立して実装できる
- ソースコードの確認は３回目
  - １回目：変更要求仕様書（＋TM）作成時
  - ２回目：変更設計書作成時



実装の生産性
80～130行/H

- 実装から人の投入が可能
  - 当初の見積りより変更規模が増えても、変更設計書により、
    実装の段階で人を投入することが可能

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

3.3.5 公式文書の作成

- 派生元の公式文書はテスト終了後に変更する
  - 開発中に作成した文書（３点セット 他）とマージする
    - テストによって変更が正しいことを確認している
    - 開発は ”派生元の公式文書” と ”差分情報” で進める
  - この方法により「並行開発」や「五月雨開発」が可能となる

公式成果物	変更要求仕様書	変更設計書
機能仕様書	●	
画面操作仕様書	●	
制御仕様書	●	
各段階の設計書（仕様書）	●	
関数仕様書	●	●
関数設計書		●
データ仕様／設計書	●	●

DENSO

4. いつやるか？ 今でしょ！

- 不具合対応を活用する
- 1人プロジェクトをチャンスに！
- 特区を作る
- ３つの質問

DENSO

4.1 不具合対応は絶好の機会！

- 不具合修正は典型的な派生開発
  - 変更要求、理由は明確、変更仕様もわかりやすい
  - テストフェーズなら、まだソースコードも記憶にある
  - 評価しやすい ・・・ デグレが出なければOK
- XDDPの手応えを得るチャンス
  - 不具合対応の小修正は、実は最も小さな派生開発
  - 変更要求仕様書 or TM だけでも書いてみる
  - 影響範囲が広い不具合修正には、XDDPの効果が出やすい
- 実際にやってみないと本当に理解はできない
  - 不具合対応だけでなく、ちょっとした修正はいくらでもある

4.2 1人プロジェクトをチャンスに！

- 1人プロジェクトの不安を自信に
  - 変更規模によっては担当者は一人 → XDDPの試行に！
  - 「思い込み」と「勘違い」をXDDPで解消
  - 周りよりうまくいっていればOK
- 1人プロジェクトの結果を盾に展開
  - 自分の担当範囲だけでもXDDPに取り組んでみる
  - 成果を社内に技術発表会やホームページで発表してみる
- 南部さんの事例 ［派生開発カンファレンス2012 最優秀発表賞］
  - XDDPの導入の挫折 から、1人プロジェクトで手応えを得た後、プロジェクトでの実績を事例発表会で発表、社内で正式な実施が決定

（「現場からの障壁克服 –***XDDP*** を導入する際の障壁とその克服に向けたアプローチ –」より ）

DENSO

4.3 特区を作るのも一つの手！

- 一つの開発チームでXDDPの試行してみる
  - 実際にプロジェクトに適用してみて、XDDPを評価をする
  - 試行の結果からXDDPをより深く理解し、次のステップを検討する
  - プロジェクトの結果を報告し、社内の展開につなげる
- デンソーでの展開事例　［派生開発カンファレンス2011］
  - 関心ある人がいるチームのトライアルからスタート
    - 成功事例を作る
    - データを取る
    - 問題の把握

⇒

- 経営層への説明
- 社外発表

（「*XDDP* の開発現場への展開 –日本的 *OJT* による導入障壁の克服 –」より ）

4.4 3つの質問

質問１ ： 派生開発は新規開発よりもやさしい？

- 要求の多様性、プロセスの問題、部分理解、影響範囲の特定 ・・・

派生開発は新規開発よりも、はるかに難しい開発です

質問２ ： ドキュメントがないからXDDPは適用できない？

- スペックアウト、設計技術、完全無知見プロジェクト ・・・

スペックアウトにより設計書を生成して進めます

質問３ ： 一人ではXDDPに取り組むことはできない？

- 不具合対応、1人プロジェクト、特区で試行 ・・・

XDDPに取り組むチャンスは身近にたくさんあります

DENSO

いつやるか？

今でしょ！

参考文献

- 清水吉男：「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，2007
- 古畑慶次：派生開発におけるプロセス構築　～*XDDP* からアーキテクチャ再構築へ ～，*Embedded Technology* 2012 スペシャルセッション 派生開発の問題解決セミナー
  - http://www.xddp.jp/tech_documents/et2012special_c7.pdf
- 古畑慶次他：*Process Improvement using XDDP - Application of XDDP to the Car Navigation System -*，*5thWorld Congress for Software Quality*，2011
  - http://www.xddp.jp/tech_documents/5wcsq_paper_kobata.pdf
- 加藤由之他：*XDDP* によるソフト派生開発の*QCD*向上活動，ソフトウェア品質シンポジウム 2008
  - http://www.juse.or.jp/software/pdf/sqip2008/ippan_06_1.pdf
- 中井栄次他：無知見プロジェクトに対する *XDDP* の適用，ソフトウェア品質シンポジウム 2009
  - http://www.juse.or.jp/software/83/attachs/ippan_2-2.pdf
- 津田剛宏他：設計手法を活用した変更要求仕様書の作成手法，ソフトウェア品質シンポジウム 2010
  - http://www.juse.or.jp/software/202/attachs/b1-3.pdf
- 南部妙水：現場からの障壁克服 -*XDDP* を導入する際の障壁とその克服に向けたアプローチ -，派生開発カンファレンス 2012
  - http://www.xddp.jp/conference2012/xddp2012_p6.pdf
- 古畑慶次：*XDDP* の開発現場への展開 -日本的 *OJT* による導入障壁の克服 -，派生開発カンファレンス 2011
  - http://www.xddp.jp/conference2011/xddp2011_P6.pdf

DENSO