変更要求仕様に時間はかからない

アフォードフォーラム

第1回講演資料

２０１１．１．３１

於：横浜市開港記念会館

（一部ページを追加しています）

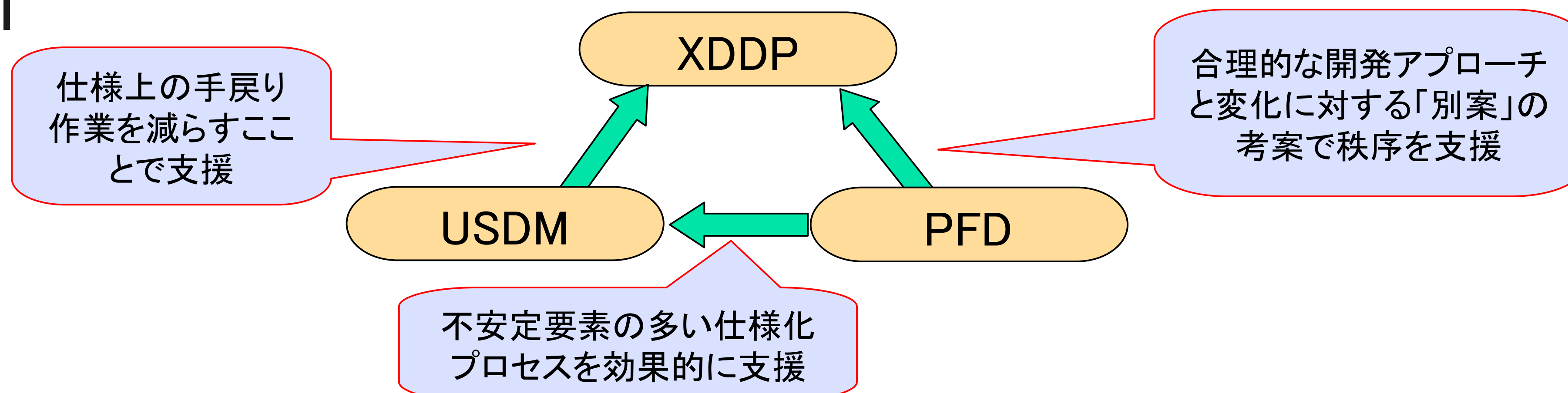
派生開発推進協議会
代表　清 水　吉 男
URL=www.xddp.jp

©Copyright 2011 派生開発推進協議会., All rights reserved

追加

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

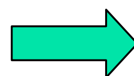


- 派生開発は短納期であったり**制約が多く条件が厳しい**
- 「USDM」
  - 追加機能の仕様モレを減らしたり、変更箇所（変更仕様）の抽出モレを減らすことで、短納期などの制約の中での開発作業を支援する
- 「PFD」
  - ムダのない**合理的な開発アプローチを設計**し「計画書」に繋げる
  - **途中で生じる変化**に対して適切にプロセスと成果物を調整する

©Copyright 2011 派生開発推進協議会., All rights reserved

XDDP,USDM,PFDの取り組み状況アンケート結果

- 2010年12月に実施
- 回答・・・22件



毎年、調査します

	XDDP	USDM	PFD
取り組んでいない	10	9	10
うまく取り組めていない	9	8	8
うまく取り組めている	3	5	4

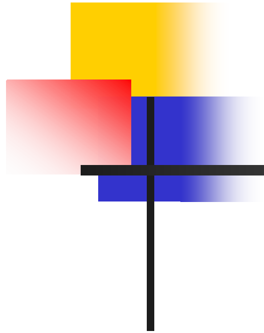
約70%が同じ組織

XDDPの3件はUSDMもできている

- 取り組めない理由（抜粋）
  - 現状を問題とは思っていない
  - 変化に対する現場の抵抗（特にXDDPの場合）
  - XDDPトライアングルが正しく理解されていない・・・効果の遅れ
  - 偽装工夫に気がついていない

質問等に関しては、HP上に「Q&A」のページを設けてお知らせします

©Copyright 2011 派生開発推進協議会., All rights reserved

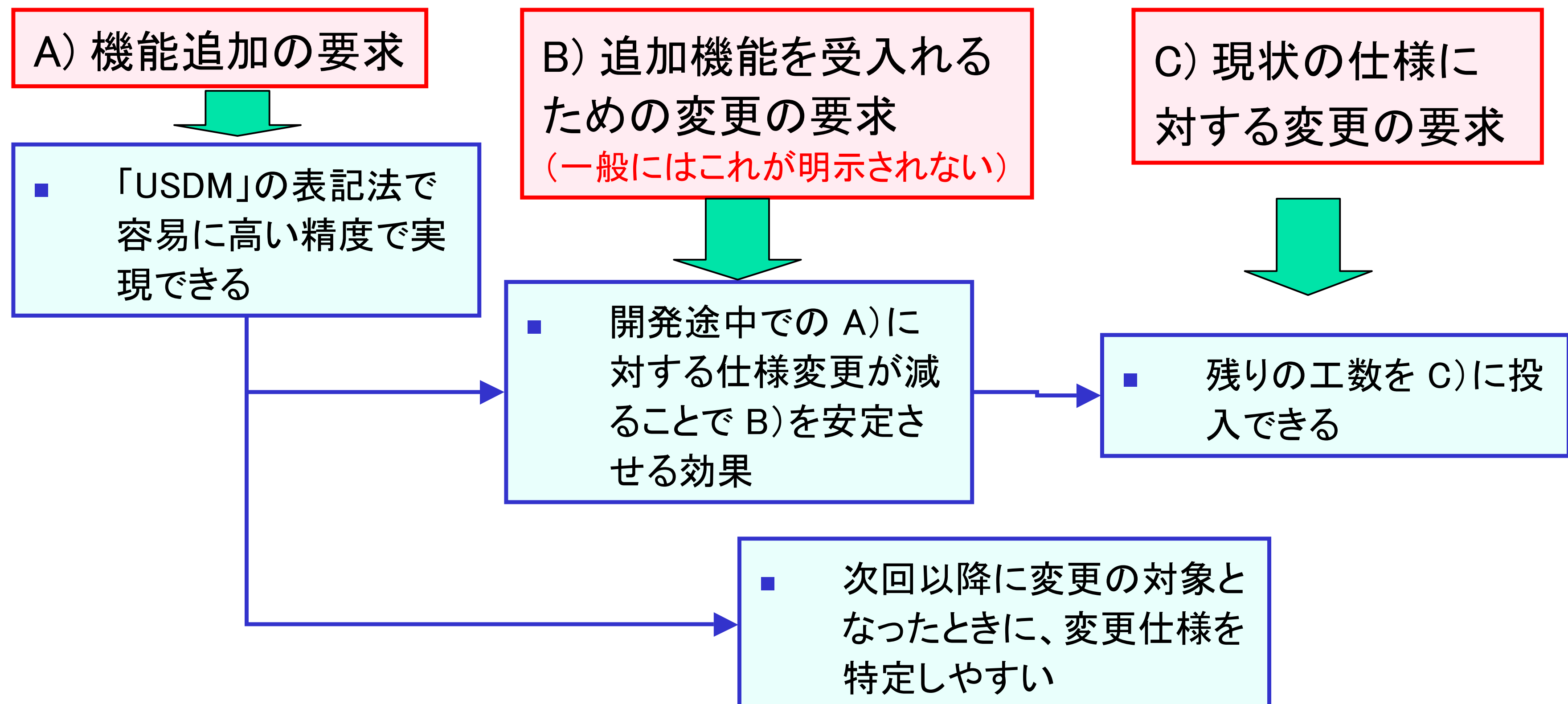


派生開発の特徴と実態

©Copyright 2011 派生開発推進協議会., All rights reserved

派生開発の特徴 (1)

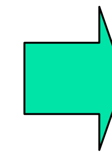
- 派生開発には3種類の要求がある
  - 追加の要求仕様書をうまくまとめることも重要
  - 変更の影響箇所を特定するのが難しい



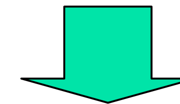
©Copyright 2011 派生開発推進協議会., All rights reserved

派生開発の特徴 (2)

- 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
  - ソースコードを読んで理解するしかない
  - 保守性を無視して作られたソースコード
  - これまでの変更で無節操に弄られていてきた
  - 読解技術も時間も不足している



全体を理解できる状況ではない



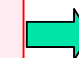
- 担当者の「思い込み」「勘違い」を防ぐことはできない
  - 担当する範囲によっては「そこに変更すべき箇所があること」にまったく気づかない
  - 従来の方法では、お互いに気づいている箇所を確認する方法も無い
  - 適切な成果物を介して「レビュー」によって問題を軽減する

©Copyright 2011 派生開発推進協議会., All rights reserved

派生開発の実態

- ソースコード上で該当箇所を見つけ次第に変更していることが、時間のロスとバグの元になっている

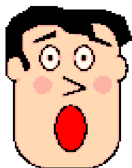
作業	作業の中の問題になる部分	時間ロス	バグ発生
仕様の問合せ／確認	個々に変更箇所を探しながら行われるため、既にソースコードを変更した箇所に影響する	○	○
変更箇所の探索	探索の過程を形にしないために、理解に時間がかかり、理解のレベルも確認できない	○	△
ソースコードになっている既存仕様の確認	機能仕様書と一致していないため、何度もソースコードを読み返すことになる	○	
既存仕様との調整／今回変更仕様の調整	理解が偏った状態で、今回の変更が既存の仕様に与える影響を適切に判断できない	△	○
前日までに変更した箇所との調整	既に変更した箇所と関連する場合、変更した理由等を思い出す必要があるが、十分に思い出せない。	△	○
変更間違いに気づいたことに伴う再変更	見つけしだいにソースコードを変更しているために、他の変更と調整はその上に積み重ねることになる		○

何よりも、より良い変更方法が見つかっても、変更し直さない  潜在バグへ

©Copyright 2011 派生開発推進協議会., All rights reserved

要求仕様書と機能仕様書の混同の問題

- 「要求仕様書」と「機能仕様書」の混同は、XDDPへの取り組みの障壁になっている
  - 新規開発で混同したまま・・・・ずっと昔から
  - 派生開発の場面で「差分」の変更要求仕様書がイメージできなくなり、ベースの要求仕様書を直接変更してしまう・・・・「新規開発崩し」
  - 「before/after」の記述ができないので、影響箇所の気付きに制限をうける

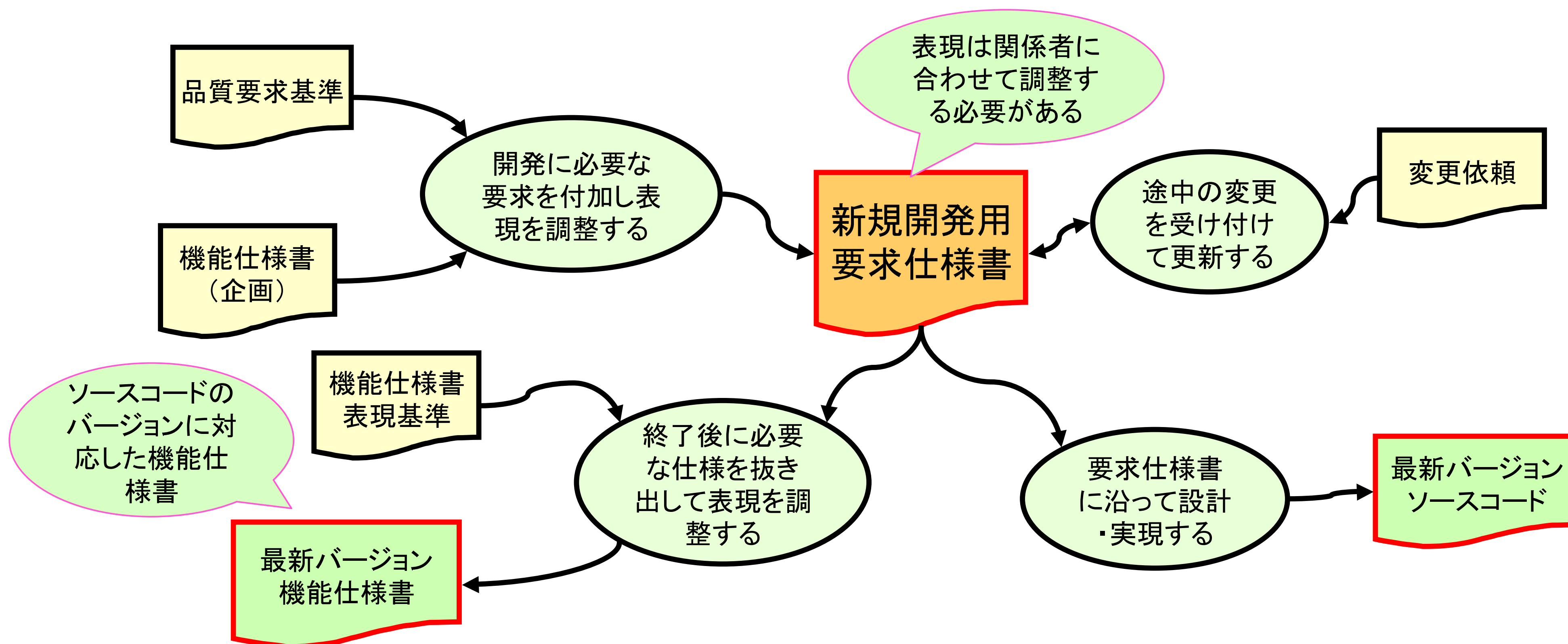


	要求仕様書	機能仕様書
目的	関係者に作ってもらうためのもの 「作り方」の要求も入る	機能を説明するもの
関係者	計画書で特定されている	特定されていない
表現	関係者が Specify できる状態でよい	一般的な記述になる
納期やコスト	背後に背負っている	意識されない
バージョン管理	今回だけの文書でとどまる	公式文書であり、ソースコードと対でバージョンアップする

©Copyright 2011 派生開発推進協議会., All rights reserved

要求仕様書と機能仕様書の関係（新規開発時）

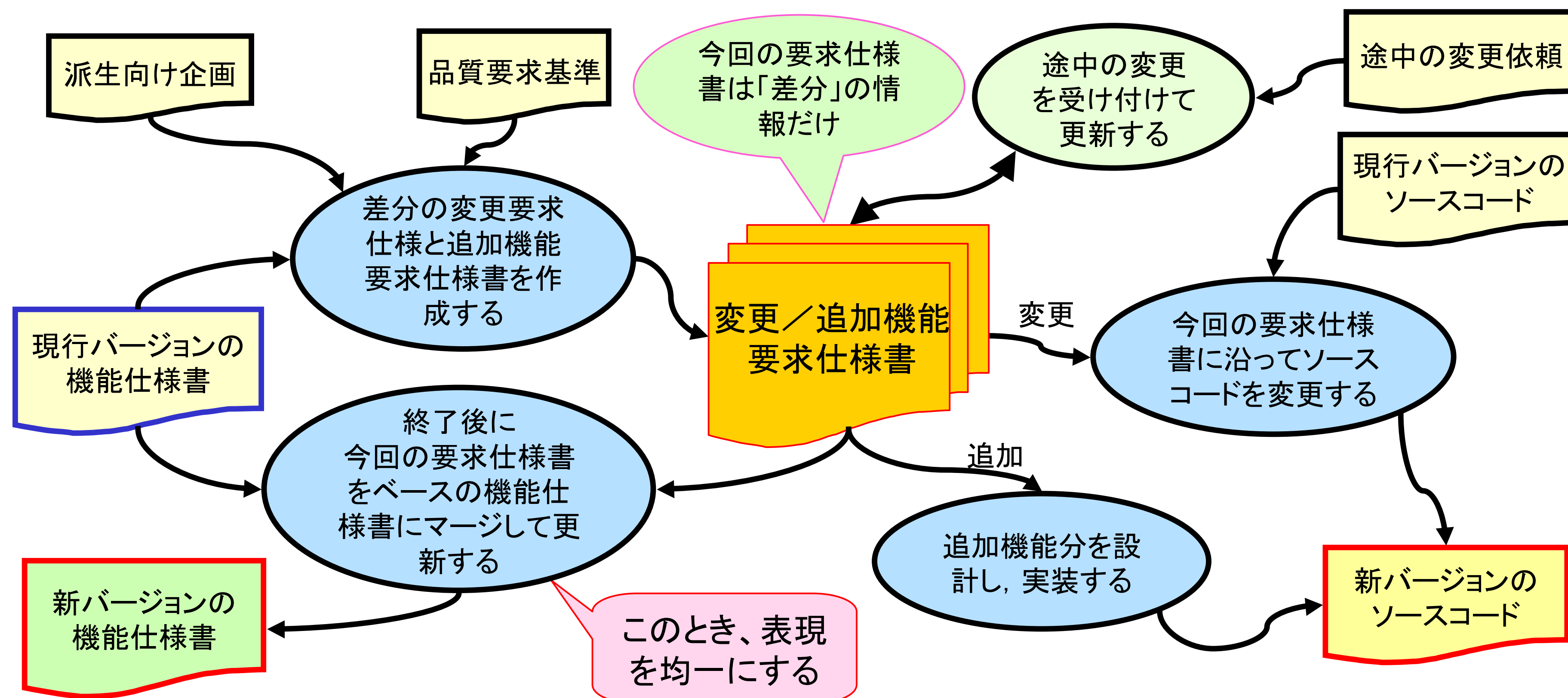
- 「機能仕様書」は提供する機能を一般的に記述したもので、関係者を特定できないために詳細さは概ね均一になる.
- 新規に“作る”ときに「要求仕様書」に変身する
- 完成後に機能仕様書に戻り、以後のバージョンアップに対応する



©Copyright 2011 派生開発推進協議会., All rights reserved

要求仕様書と機能仕様書の関係（派生開発時）

- 派生開発での要求仕様書は「変更要求仕様書」と「追加機能要求仕様書」に分かれ、2つの要求仕様書（差分）に基づいて開発作業を進める
- 完成後に、これを元にしてベースの機能仕様書を更新（マージ）する



©Copyright 2011 派生開発推進協議会., All rights reserved

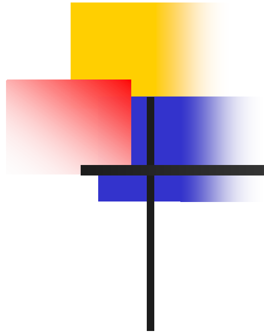
要求仕様書と機能仕様書を使い分けよう

- 対応策
  - 現状の「要求仕様書」のタイトルを変えて「機能仕様書」として扱う
  - 以降の派生開発から「差分」の「変更要求仕様書」を用意する
  - 文書の構成が違っても工夫次第（従来は「ward」でつくられている？）
    - 今回変更した関連部分に限定して「USDM」形式に書き直す
    - 追加機能は、該当箇所に「別冊参照」を挿入して対応する

「できない言い訳」を考えるのではなく、「できる方法」を考える

これでもいいじゃないか。
よくなるのだから。

©Copyright 2011 派生開発推進協議会., All rights reserved



変更要求仕様の話

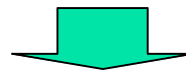
 ©Copyright 2011 派生開発推進協議会., All rights reserved

「変更要求仕様書」を導入する

- 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

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<u>変更内容まで含めて</u>その内容について特定（Specify）したことがまとめられた文書

- 変更部分だけを「差分」で表現する
- 「変更要求」と「変更仕様」を階層関係で捉える
- すべて「before / after」で表現する
- 変更プロセスには、いわゆる「設計プロセス」が存在しないので、ソースコード（関数仕様）のレベルで変更仕様を記述することで、この段階で実現方法まで特定する（Specify）

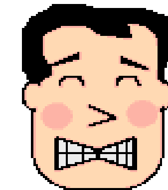


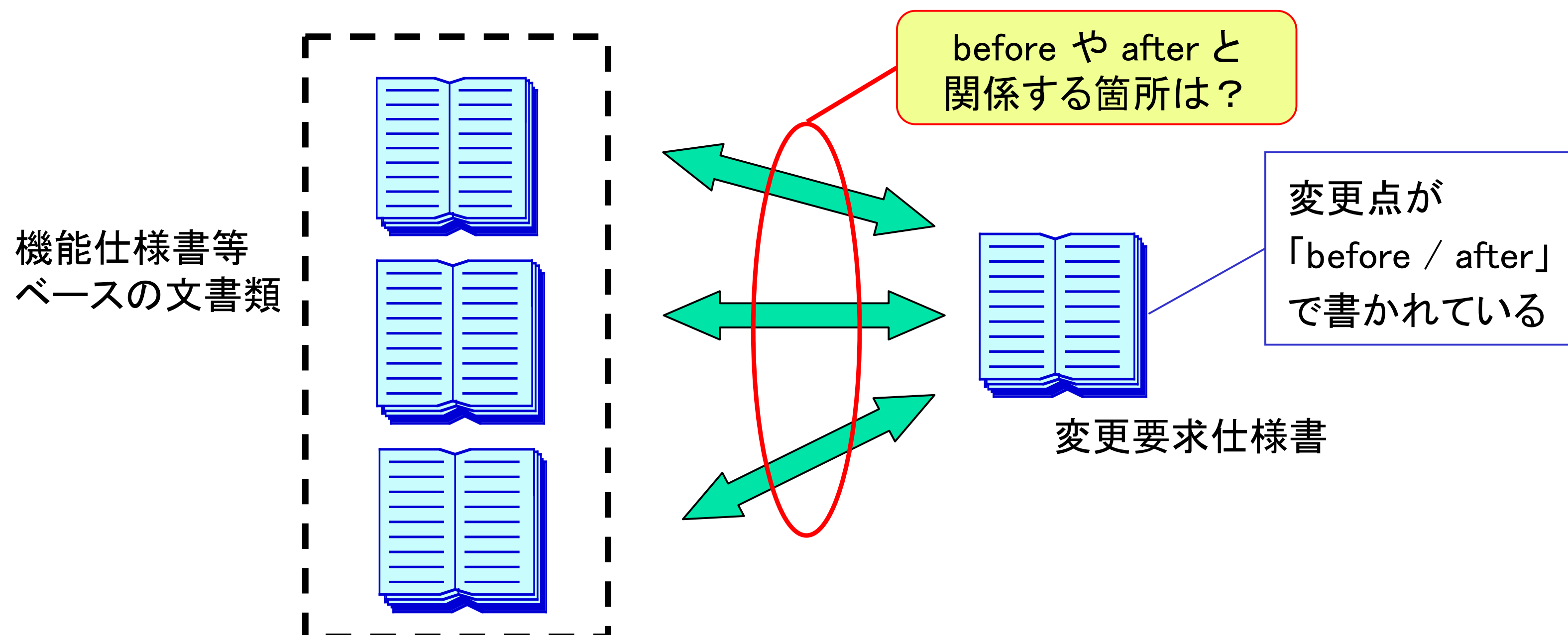
終わってから「機能仕様書」などの公式の関連文書にマージする

©Copyright 2011 派生開発推進協議会., All rights reserved

「差分」だけで大丈夫？

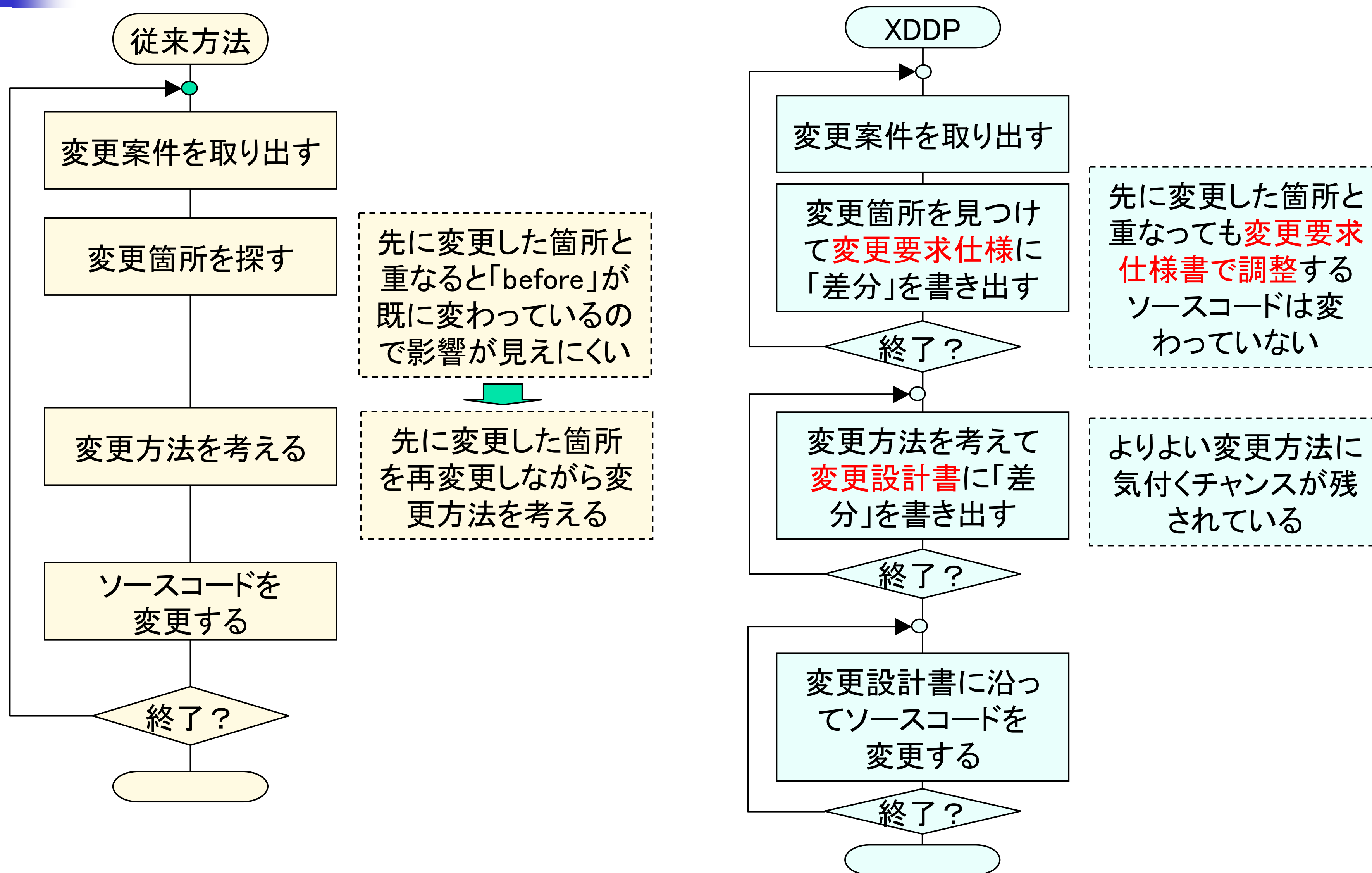
- 変更後の様子は？
  - ベースの機能仕様書の構成に沿ってまとめる
  - 変更後の様子は、ベースの機能仕様書と並べて、変化する箇所を見ればよい
- 「差分」の効果
  - 「before」･･･これと共有したり関係する仕様は他にないか検討しやすい





©Copyright 2011 派生開発推進協議会., All rights reserved

XDDPにおける変更作業の流れの違い

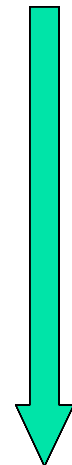


©Copyright 2011 派生開発推進協議会., All rights reserved

変更要求を立てるには工夫が必要

- 一般に変更の依頼は「仕様」のレベルで出される

	□ □ □		家族で利用料金をまとめて20%割引いて欲しい



- 依頼された箇所は変更されるが、変更箇所を間違えるたり、変更すべき箇所が足りないことが起きる
- 理由を探って変更要求を捉える・・・・最重要項目
  - 変更の理由は？　…他社との競争に勝つため
  - そもそも何が変わるのか？…一律の割引では効果が薄い！

要求	加入者データに家族データの定義を追加して、人数が多い程割引率を高くなるように変更する
理由	関係者を家族として抱え込むことで他社に乗り換えにくくする

- 変更要求を捉えることで
  - 必要な変更箇所を漏らさない
  - 変更方法を間違えない

©Copyright 2011 派生開発推進協議会., All rights reserved

変更箇所を＜グループ＞で捉える

- 変更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

要求	CCL30	加入者データに家族情報を追加し、人数が多い程割引率を高くなるように計算を変更する
	理由	関係者を家族として抱え込むことで他社に乗り換えにくくし、経営を安定させる

- 変更箇所は･･･変更要求の中の作業者が行う「動詞」がヒント
  - 加入者データ：データ項目の追加が必要
  - 加入者の表示：加入者を表示している画面で「親」と「子」の関係での表示が必要
  - 計算：家族の人数を把握する必要がある
  - ：人数に応じて割引率を選択する
  - ：集約した先を各加入者データに残す　→データ項目に必要
- 変更の範囲を＜グループ＞で記述する

		＜加入者データの追加＞
		＜加入者データの表示の変更＞
		＜計算方法の変更＞

©Copyright 2011 派生開発推進協議会., All rights reserved

変更仕様に書き出す

- 「変更要求」や、変更の範囲を示す<グループ>を参考に、関係箇所を探す
  - 機能仕様書
  - 各種設計書
  - ソースコード　など
- 最終的に、ソースコード上の該当箇所を見つけて、変更要求仕様書に「変更仕様」として書き出す

  ① 設計書などを参考にする（存在していれば）

  ② <グループ>をキーにしてソースコードを読んで（スペックアウトしながら）探す

  ③ 該当箇所が見つかれば、その部分を「before/after」の形にして、（変更要求の下に）変更仕様として記述する

  ④ 関数が特定できていれば、TMの該当するセルに関数名を記述する

> 今まで・・・その場でソースコードを変更していた
>
> これから・・変更要求仕様に書き出す

©Copyright 2011 派生開発推進協議会., All rights reserved

変更仕様を＜グループ＞でまとめる

- 変更仕様・・・この変更要求で実際に変更する箇所。変更の影響範囲が見える
- TM（Traseability Matrix）・・・変更箇所が存在する場所が見える

				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	以下の10個の家族データを追加する ・主従区分（主＝1、従＝3～8） ・加入者数（区分＝従のときは「空」） ・加入者番号（区分＝従のときは「1個」だけで「主」の番号を記載		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 （ケース毎の割引率一覧）				f9()

©Copyright 2011 派生開発推進協議会., All rights reserved

変更仕様は箇所を特定できるように表現する

- 仕様の変更や削除は「before」を表現することで該当箇所を特定できる

場面　before　after

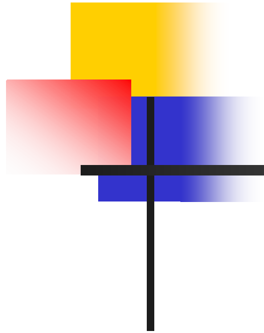
	□ □ □	LIB20-03	インターネット貸出し操作時に「貸出し中」にセットしているのを「予約中」に変更する
	□ □ □	LIB30-02	検索画面で表示される書籍情報の「版番号」の表示を削除する

- 仕様の「追加」は、「before」が存在しないので、追加・挿入する箇所がわかるように表現する必要がある

挿入する箇所を示す

	□ □ □	LIB20-03	パスワードの入力の前に社員コードから現在貸出し冊数をチェックして、10冊のリミットに達しているときは下記の警告を表示して以降の操作を拒否する処理を追加する
	□ □ □	LIB30-02	検索画面で表示される書籍情報の書籍名の下に「痛み具合」の表示を追加する 痛み具合区分：0　表示しない 痛み具合区分：1　“少々傷んでいます” 痛み具合区分：2　“かなり傷んでいます”

©Copyright 2011 派生開発推進協議会., All rights reserved

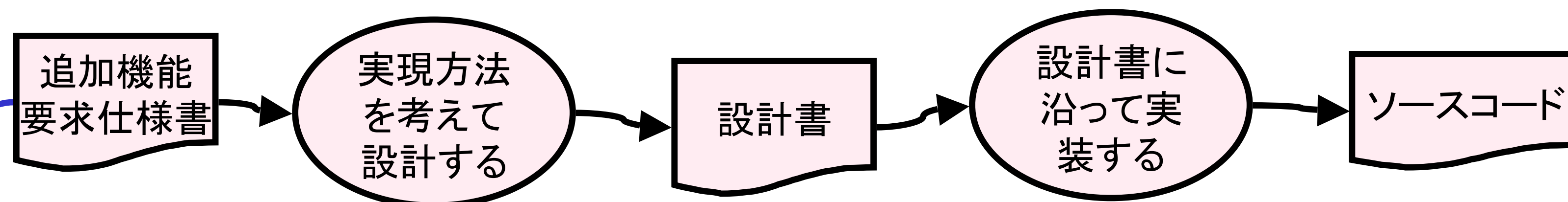


追加機能受け入れの変更要求仕様の話

©Copyright 2011 派生開発推進協議会., All rights reserved

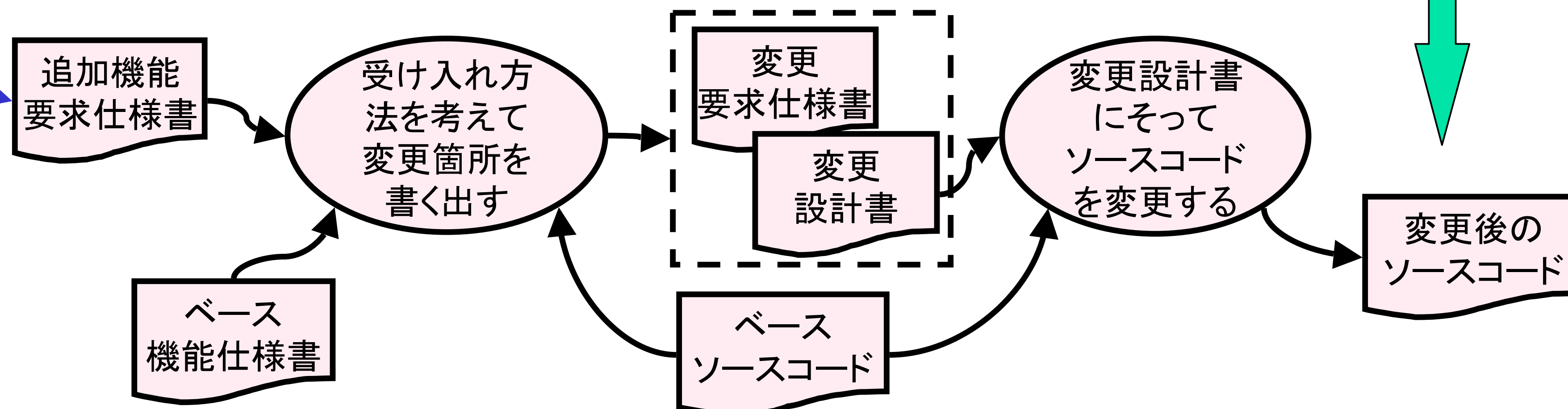
機能追加と受け入れの変更がペアになる

- 追加機能
  - 追加機能要求仕様書に基づいて「設計」し、ソースコードを作成する



受け入れ

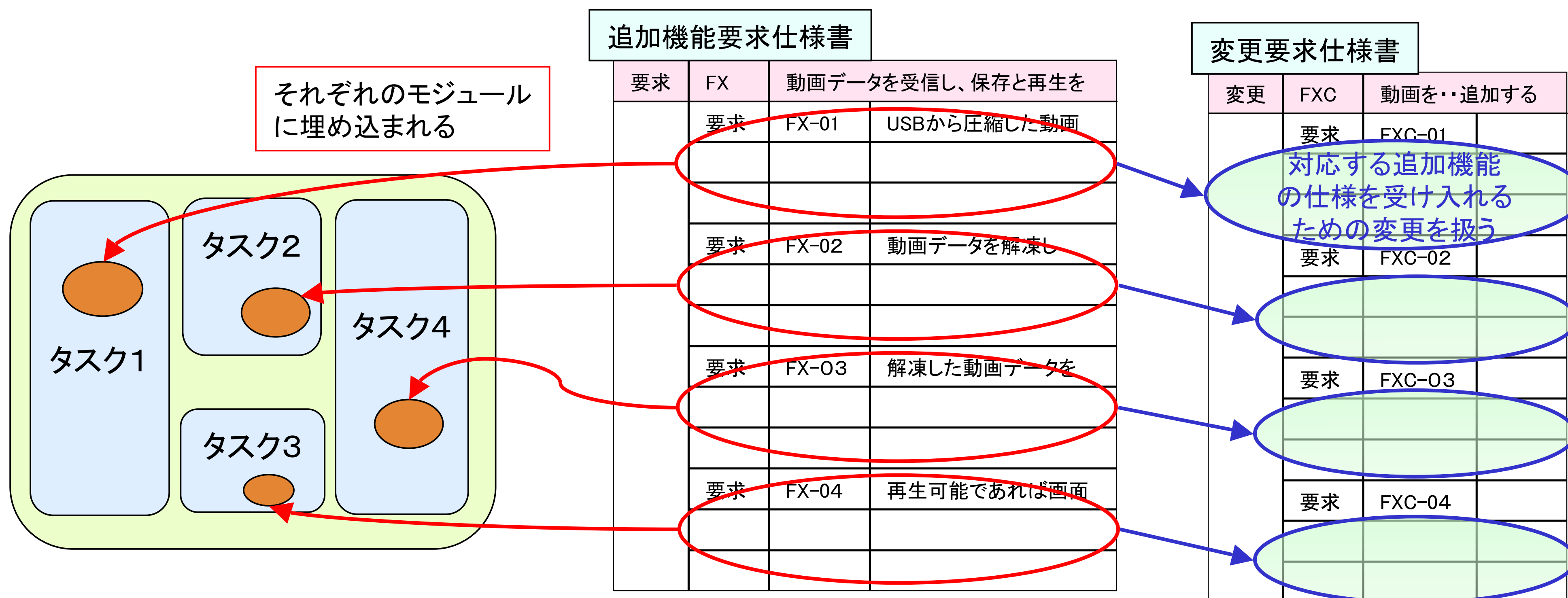
- 変更
  - 追加機能要求仕様書を調べて、この機能（仕様）を受け入れるためにベースのソースコードを変更する必要がある



©Copyright 2011 派生開発推進協議会., All rights reserved

追加機能要求仕様書のポイント

- 新規開発時との違い・・・システムのアーキテクチャが存在している
  - 追加される機能が、複数のタスクなどに分散して埋め込まれるときは、2層目の要求をアーキテクチャに対応させる
  - 受け入れの変更も追加機能の要求仕様書の構成に対応させることで、追加機能が作動する状況を確認しやすくなる



©Copyright 2011 派生開発推進協議会., All rights reserved

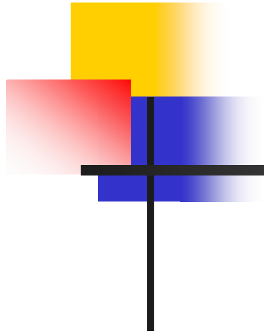
追加機能を受け入れるための変更

- 追加機能を受け入れのための変更仕様
  - ベースのソースコードから新しい処理を呼ぶための変更が中心になる
    - 画面上に新しいボタンを追加して、それを押すことで新しい関数を呼ぶ

      「条件を判断して、新しく作成された関数を呼ぶ処理を追加する」
  - 追加される機能の動作環境を確保するための変更もある
    - 既存の機能から未使用のメモリースペースを集める変更
    - 動作タイミング確保するために既存処理の実行間隔を広げる変更
    - 全体の処理時間を維持するために、既存機能の処理時間を短縮する変更

- 追加機能要求仕様書から変更要求仕様書に移るケース
  - 追加機能の仕様として記述していたが、既存のソースコードを変更することで対応できる場合は、変更要求仕様書に移す
    - 各種の設定、画面の操作や表示に多い
    - 追加機能要求仕様には、変更要求仕様に移動したことをメモる

©Copyright 2011 派生開発推進協議会., All rights reserved



工数と効果の話

 ©Copyright 2011 派生開発推進協議会., All rights reserved

何の工数が増えて、何の工数が減るのか (1)

- 増える工数
  - 変更要求仕様書に書き出す
    - 但し、従来も変更すべき内容を認識した上でソースコードを変更しているので、増えるのは「before/after」の形式で書き出す工数のみ
    - TMに関数名を書き出す作業は工数的には微々たるもの
    - 逆に、変更要求の下に書き出した効果として
      - 他に関連箇所(影響箇所)に気付きやすくなる
  - 変更設計書に具体的な変更方法を書き出す
    - 従来も、挿入箇所を含む具体的な変更方法を考えついているので、増えるのは文章で書き出す工数のみ
    - 逆に、書き出した効果として、
      - 変更方法を見直すことができる
  - “不慣れ”に起因する工数
    - 文章で書き出そうとするとなかなか言葉が出てこないことが起きる
    - 逆に、文章にならない状態でソースコードを変更することは危険

©Copyright 2011 派生開発推進協議会., All rights reserved

何の工数が増えて、何の工数が減るのか (2)

- 減る工数
  - 他の変更要件に対応しているときにいろいろなことが起きるが、変更内容や変更方法が記述されていることで、工数が削減される
  - 思い込みや勘違いを軽減するレビューが可能となり、バグが大幅に減少する

重要なことは、XDDPではまだソースコードは変更されていない

場面	XDDP	従来方法
既に変更した箇所と変更が重なることが判明	ソースコードと書き出された変更情報と見ながら、最適な変更方法を考え出せる。	ソースコードは既に変更されていて元の状態が見えないまま、新たな変更が加えられる。
2週間前に間違って変更したことに気付く	ソースコードを見ながら、変更要求仕様の記述を正しい変更内容に書き換える。	
既に変更した案件の変更依頼の意図が不安になった	変更要求仕様を読み返して必要に応じて依頼者に確認して、勘違いが分かった時は新たな変更方法に書き換える	既にソースコードが変更されていることで、読み返してそのときに考えたことを思い出そうとしても容易ではない。時間がかかるし、不確実。
10日前に変更した箇所が気になる	変更要求仕様を読み返して確認する。ソースコードは元のままなので、3日前の状態を再現しやすい。	
もっと効果的な変更方法が見つかった	ソースコードはまだ変更されていないので、変更要求仕様や変更設計書を書き直せばよい。	「あの変更でも間違っていない」として不問に付す。

©Copyright 2011 派生開発推進協議会., All rights reserved

手戻り工数の削減

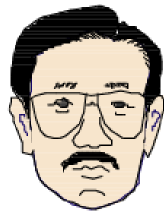
- 派生開発では、バグを仕込まないことが重要になる → コードの劣化防止も
- 「３点セット」によって、多くのバグはソースコードを変更する前に発見される
  - 「What」「Why」「Where」「How」の視点の効果

問題	変更要求仕様書	TM	変更設計書	対応
他の変更箇所に気付かなかった	○	○		関数レベルの変更仕様とTMによって担当者が変更しようとしている方法と箇所が見える
同じ関数で変更箇所が複数あった	○		○	変更設計書で具体的な変更箇所を文章で記述することで、変更箇所の不足に気付く機会を提供
依頼内容を勘違いして変更した	○			変更要求や理由を記述することで勘違いに気付きやすい
別の担当者と変更箇所が重なった	○	○		TMで同じ関数にぶつかっていることが分かり、この段階で調整できる
			○	別チームとの変更箇所の衝突は、先行チームのソースコードの変更を待って調整する
不適切にデータ構造を変更したことで他の機能に影響を与えた	○			変更要求と変更仕様の階層構造、さらに変更の範囲をグループで表現することで気付きの機会を提供

©Copyright 2011 派生開発推進協議会., All rights reserved

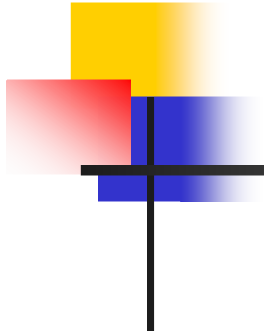
変える勇気を！

- 「XDDP」は、派生開発におけるムダな作業を徹底的に取り去った先にある開発アプローチ
  - 上手に取り組むことで大きな効果が得られます
- 「仲間」がいることで勇気をもらうことができます
- 市場の要求が変化しているところで、変化を拒めば市場から見放されます



取り組み結果をこの場に持ち寄って、
ヒントと勇気を持って帰ってください

©Copyright 2011 派生開発推進協議会., All rights reserved



「保守性」の表現について

©Copyright 2011 派生開発推進協議会., All rights reserved

要求仕様書との関係

- 「保守性」・・・作り方の品質の代表格
  - 作業者に対する「指示」となる
    - 「・・・のように作って欲しい」
    - 「・・・の作り方はしないで欲しい」
  - 要求仕様書は本来全てが作業者への「制作依頼書」であり、その中で「作り方」の要求が表現できる
  - 「要求仕様書」と「機能仕様書」を使い分けることで、明確な「保守性」の表現が可能になる
- 最近の傾向として、「Maintainability」が「Supportability」や「Serviceability」に置き換わる動きがある
  - 機能仕様書の発想では「Maintainability」は書けない
  - 「Maintainability」が要求仕様書から外されれば、その後の派生開発に大きな影響がでる
    - “作りっぱなし”のシステムに限定すべき

©Copyright 2011 派生開発推進協議会., All rights reserved

機能追加における「保守性」の品質要求

- 今回追加される機能に対する品質要求は、新規開発時の品質要求と基本的には同じ
  - 機能を補足する品質要求
  - 交換性や保守性などの品質要求

要求	QUA20	[保守性]追加機能の保守性を既存の保守性のレベルを継承して欲しい	
	理由	システム全体の保守性のレベルを落としたくない	
	□ □ □	QUA.02-1	モジュールの複雑度は20以下を原則とする
	□ □ □	QUA.02-2	“手順的凝集度”以下になる場合は事前に承認をとる
	□ □ □	QUA.02-3	処理と管理は明確に分離し、関数の呼び出しの深さは5以内とする
	□ □ □	QUA.02-4	タスク間でアクセスし合うグルーバル・データを作らない. グローバル・データとなることが避けられない時は、事前に承認をとる. 【説明】割り込み処理との間で共有するケースはこの制限外となる.

これらの仕様が実現すれば、
保守性は維持されているとみなす

©Copyright 2011 派生開発推進協議会., All rights reserved

変更に対しても「保守性」の品質要求を課す

- 変更における保守性の要求は「劣化」を防止するのが目的であり、不用意なソースコードのコピーを禁止したり、凝集度や複雑度の悪化を防ぐための （限定的）リファクタリングの基準を設けたりする
- 機能追加分の保守性の要求とは別に用意する

要求	QUA20	[保守性]変更の際に現状の保守性のレベルを悪化させない	
	理由	もともと「保守性」を考慮して設計されているから	
	□ □ □	QUA20-01	処理内容の一部を不用意に複製せず共通機能分割で対応する 【説明】他の呼び出し箇所を変更することになる
	□ □ □	QUA20-02	変更によって凝集度を悪化させる場合は、モジュールを適切に分割して独立させる（制限付きリファクタリング） 【説明】この場合、新しいモジュールの仕様／設計書が必要
	□ □ □	QUA20-03	新たな処理を組み込むことで複雑度が20を超えるこきはモジュールを適切に分割する 他に方法がない時は、事前にPLの許可を得ること
	□ □ □	QUA20-04	新たにタスクの外にグローバルデータを出さない 他に方法がない時は、事前にPLの許可を得ること

©Copyright 2011 派生開発推進協議会., All rights reserved