System Creates

PFD(Process Flow Diagram)の書き方

第3版

2009/9/21

提供：(株)システムクリエイツ

代表取締役 清 水　吉 男

URL=http://homepage3.nifty.com/koha_hp

内 容

1. PFDの紹介
2. PFDの書き方（基本）
3. PFDの書き方（応用）

1. PFDの紹介

- 私がPFDを使ってきた理由や、PFDの利用方法について説明します。

1. PFDの由来
2. プロセスを固定した過ち
3. 品質を確保する2つの原則
4. プロセスを自在に設計するスキルの意義
5. CMMで価値に気付く
6. 組織標準とテーラリング
7. 定義から設計へ
8. ツールについて

1.1 PFDの由来(1)

- 私自身は、1970年代後半に構造化設計に取り組んだ辺りから、構造化分析のツールである「DFD(Data Flow Diagram)」を、ソフトウェアの開発作業はもちろん、その他のあらゆる作業の表現に利用してきました。
- しかしながらDFDの場合は、「成果物」の表現が「ストア」記号を使用したり、フロー上の「データ」で表現したりと明確な基準がありません。
- これでは、慣れない人には書きにくく、読み手にとってもイメージしにくいために、「文書」の記号を導入したところ、非常に分かりやすくなったのと、プロセスは必ず成果物を挟むというイメージが掴みやすくなりました。
- そこでこの表記法を説明する際に「DFD」と読んだのでは、構造化分析のDFDと混乱するので、「PFD」と呼び方を変えたのが始まりです。

1.1 PFDの由来(2)

- 「PFD」は、それまでの「DFD」に対して「文書」の記号を導入したのと、成果物にも番号を付けることを導入した以外は、DFDと同じですので、これまでDFDを使用してきた方であれば、すぐに理解できるでしょうし、DFDと同じように使うことができるでしょう。
- また「PFD」そのものは非常に直感的ですので、初めて「PFD」を書こうという人も、そこで使われる「記号」や「ルール」はすぐに覚えることができ、何度か書いているうちに慣れると思います。
- 「PFD」は単なる「図」ではありません。思考の道具です。これを使ってイメージしたことを自在に表現するスキルは、問題を前にして解決策を柔軟に引き出すはずです。

1.2 プロセスを固定した過ち(1)

- 日本では、過去に何度か「プロセスを固定」する過ちを犯しました。
  - ソフトウェアの世界にエンジニアリングの考え方を持ち込んだとき
    - 製造の世界との違いを認識せずに誤って固定しました。
    - ソフトウェアの開発アプローチで製造プロセスと共通するのは、通常は関数の実装工程とテストの工程ぐらいです。あるいは少し前の具体的に関数レベルの構造がデザインされ、関数仕様を書くあたりからです。そこまでは「分析」や「設計」の世界であり、このあたりのプロセスは製造の世界とは性質を異にします。
  - 「ISO-9001」を導入したとき
    - 「ISO」の誘導ミスかと思われます。
    - 製造プロセスとの違いを認識しないままに取り組んだことも、「固定」に繋がったと考えられます。
  - 「CMM」を導入したとき
    - 「ISO-9001」の延長線でCMMを捉えたためと思われます。
    - 「プロセスは固定させるもの」という考え方から脱却できていません。

1.2 プロセスを固定した過ち(2)

- 「固定」であっても、プロセスを考えた時点の市場の要求を反映していますので、しばらくは対応できるはずです。
- しかしながら、90年代に入って市場の要求が変化(納期の短縮要求)したときに応えることができなくなったのです。
  - 本来取られるべき対応策は、プロセスを変化させて一人の生産性を高めることでした。
  - 実際には、個人の分担を狭くしたり、担当を固定させたりして、必死にカバーしようとするものの、プロセスを設計できるスキルを確保してこなかったために対応できません。
  - そして90年代後半には、さらにコストの削減要求が出されたのを機に、今度はオフショア作戦で逃げたのです。プロセス面では何も解決していません。
- 市場が求めているのは、絶えざる「変化」です。
  - 機能が変化し、性能が変化し、期間もコストも変化しつづけます。
  - 変化する要求に応えるには、変化をプロセスに取り込む必要があります。

1.3「安定」させればよい

- プロセスを「変化」させることは「品質」の面では大きなリスクです。
  - 確かにプロセスを固定しても一時的には品質を確保できますが、市場の要求が変化する以上、固定したプロセスはどこかで要求に応えられなくなります。
- プロセスを「固定」させるのではなくて「安定」させるという考え方を持ち込むことで、この相反する要求に応えることができます。
  - PFDは、ここで最高に威力を発揮します。
  - PFD図や成果物とプロセスの定義を使って「シミュレーション」ができます。
  - 「シミュレーション」によって合理性を欠くプロセスや曖昧な定義を修正します。
  - 何度も「シミュレーション」を繰り返すことで、「経験」のレベルに高めます。
  - そして実際の作業の場面では、既に何度も「経験」済みの状態ですので迷うことはありません。つまり「安定」の状態になっているのです。
  - さらに、今日の結果から、1ヶ月先のプロセスに影響することもわかるのです。それがPFDの効果です。

1.4 品質を確保する2つの原則

- 品質を確保するには2つの普遍的な原則があります。

① 今回の要求を実現できる合理的な開発アプローチを持つ
- この時、変化する要求に合わせて自在に設計できるスキルが必要です。

② 開発アプローチの中のそれぞれのプロセスを適切に実行できるソフトウェアエンジニアリングの技術を持つ
- プロセスを変化させたときは、技術の適用方法も変化させる必要があります。

- 開発アプローチを自在に設計できるスキルを持っていることで、開発の途中で変化する要求や状況に対してもプロセスを変化させて「別案」を考案することで対応できるのです。
- 「PFD」はこのスキルを支える重要なツールなのです。

1.5 大事なのはプロセスを自在に設計できるスキル

- 今回の「要求」を読み取って、この要求を実現するための合理的な開発アプローチを「設計」します。
  - 「要求」には、機能の実現だけでなく、納期やコストも含まれます。
  - また開発アプローチを考えるときは、自分たちの体制なども考慮します。
- 「要求」は毎回変化します。そのため、開発アプローチもこの変化した要求に応える必要があります。
  - そのためには、各自が自分の担当範囲のプロセスを自分で設計するスキルを身につけることが重要です。
  - 変化するといっても大きく変わることは少なく、大部分は類似しています。そこで参考にしているのが以前に自分で設計したプロセスであれば、合理性を失うことなく変化させることができるのです。
- 組織の一人ひとりが自分のプロセスを自在に設計できるスキルは「CMMI」のレベル4～5の組織に不可欠なスキルと考えています。

1.6 CMMでPFDの価値に気付く

- 1991年に、CMMと遭遇したとき、それまでの15年間の現役時代で、一度も納期を遅らせることなく「SEの仕事を楽しく」やってこれた原因として、「PFD」が大きな比重を占めていたことに気付いたのです。

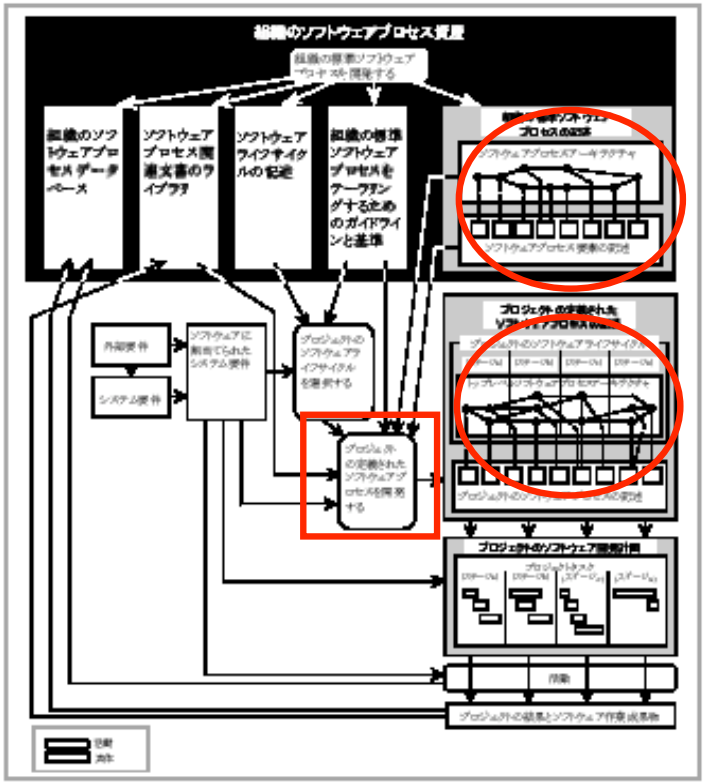


図4.1 CMMで使用される概念的なソフトウェアプロセスの枠組み

- 右の図は、CMM v1.1の中に出てくる図ですが、「○」で囲った部分は、構造化分析の「DFD」の表記を、ほとんどそのまま使用しているものと思われます。
- 言い換えれば「○」で囲ったところが、私の「PFD」と同じ働きをしているわけです。
- そして「□」で囲った部分のテーラリングの考え方も、PFDを使った私のやり方と同じだったのです。
- このように、CMMは私のポケットの中身の価値を教えてくれたのです。

1.7 組織標準とテーラリングについて

- 「組織標準とテーラリング」は、CMMやCMMIの取り組みの中で知られるようになりましたが、「DFD」や「PFD」を使ってプロセスを設計したことのない人たちに、この言葉の意味を正しく理解するのは難しいかもしれません。
  - CMMI v1.2では「選択」と「テーラリング」という言葉ができてきます。
- 選択
  - 組織標準としていくつか用意されたPFDのパターンから、今回の要求に近いものを選びます。
  - たとえば、最上位のPFDとそこに現れる基本的な成果物やプロセスの定義がセットされたもので、「新規開発用」と「派生開発用」でそれぞれ規模の違いや特徴によって数パターン用意します。
- テーラリング
  - 下位層に展開する中で、今回の要求に合わせて成果物を増やしたり、定義の中身を追加していき、「プロジェクト標準」に姿を変えます。
  - 基本的には、テーラリングは「追加」する方向に統一します。

1.8 定義から設計へ

- 一般には「プロセスを定義する」と認識していますが、「定義する」には「固定する」というイメージがつきまといます。
- その結果、「組織標準」もそのまま実際の場面に使える状態のものが用意されることになります。それでは、要求に合わせて「プロセスを自在に作る」スキルは身に付きません。
- 「定義する」からは、要求毎にプロセス（開発アプローチ）を作り替えたり、途中での変化に対してプロセスを変化させるという発想は出てきません。
- 「プロセスを設計する」と考えることで、変化する要求に対して合理的な開発アプローチを考え出すという行為につながるのです。
  - PFDによって成果物とプロセスの合理的な連鎖を設計し、
  - 成果物の構成を設計し、
  - それらの成果物を生み出すプロセスの中のアルゴリズムとシーケンスを設計します。

1.9 ツールについて

- これまでは適切なツールがなかったために、Excel で書いてきました。
  - Excel でのシート構成は以下のようになっています。
  - PFD−0、PFD−1・・・・というように「PFDシート」を必要なだけ追加します。
  - 「成果物定義シート」は、最上位のPFD（PFD−0）にすべての成果物が見えますので、その成果物を対象に定義します。（1枚のシートで構成）
  - 「プロセス定義シート」は、「PFD−x定義」というようにして「PFDシート」の単位毎に対応するようにプロセスの内容を定義します。（基本プロセスのみ）

- 今回、スパークスシステムズ ジャパンの河野さんの方で、

「EA (Enterprise Architect)」のアドインツールとして作ってくれました。

http://www.sparxsystems.jp/

  - PFDの部分をこのEAのアドインツールで作成し、成果物とプロセスの定義は、そこからExcelのシートに繋がるようになっています。

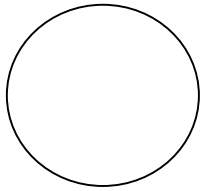
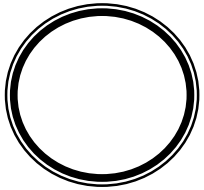
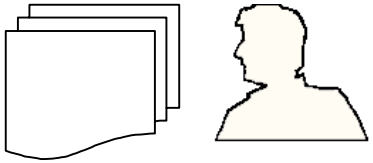
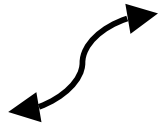
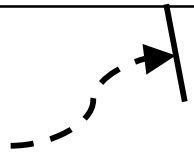
2. PFDの書き方（基本）

- PFDの表記のルールは、成果物の部品のパターンが変わっているだけで、基本的にはDFDと同じです。

1. PFDで使用する基本的な記号
2. プロセスの表現
3. 成果物の表現
4. フロー
5. 生成型と更新型
6. 階層化
7. 番号を付ける
8. 成果物の定義
9. プロセスの定義
10. 関係を表すだけ

2.1 プロセスで使用する基本的な記号

- PFDで使用する記号は以下の通りです。

	プロセス	作業を表します。 階層を持つ場合には線を太くしたり二重にすると良いでしょう	
	成果物	プロセスに出入りする成果物を表します。ソースファイルや、分冊の様子や、形になっていない「ノウハウ」など、適当な記号を使います。	
	フロー	成果物とプロセスを繋ぐ線。両側に矢印を付けて更新の意図を表したり、線上に、成果物を構成する要素を書くこともできます。	
	トリガー	プロセスの起動タイミングを表現する必要が生じたときに使用します。できるだけ使用せずに済ましてください。	

2.2 プロセスの表現(1)

- 「プロセス」は、1つの「○」(1重線)で表現します。

  - プロセス名は「目的語ー述語」で書きます。
  - 1重線のプロセスは、実際に作業を行うプロセスですので、できるだけ、動作の範囲や入力物が出力物に変換される様子がリアルにイメージできるように表現してください。
  - 「リスクの抽出」のような表現では、具体的な行動(作業)がイメージしにくいので、避けてください。
  - プロセスにおける具体的な作業は「プロセス定義」で記述します。

2.2 プロセスの表現(2)

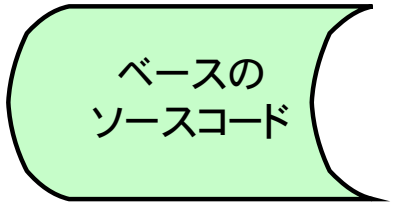
- 下位層を持つプロセス(上位プロセス)は、2重線の「◯」で表現します。

  - プロセス名は「目的語ー述語」で書きます(これは1重線のプロセスと同じ)。
  - 実際に作業を行うプロセスは、これより下の階層に現れます。
  - できるだけ、動作の範囲や入力物が出力物に変換される様子がイメージできるように表現してください。ただし、関係する成果物が多くなると思われますので、ある程度は「総体的」な表現になるかも知れません。
  - それでも「仕様書作成」「機能設計」のような表現は避けましょう。

2.3 成果物の表現

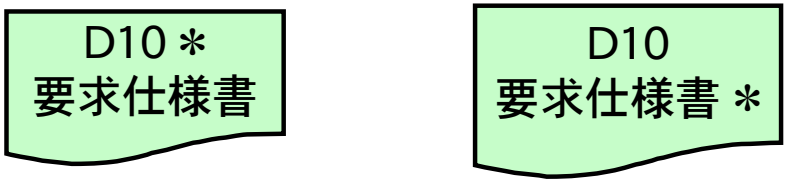
- 成果物は通常は「単票」または「複票」の部品で表し、その中に成果物の名称を記入します。
  - タスク設計書のように、何冊も存在するものをまとめて表現する場合は「復票」で表し、具体的なイメージを誘います。
  - 成果物の構成などは、別に「成果物定義」で記述します。

- ソースファイルなども「成果物」として表現します。
  - 認識しやすいように文書とは別の記号を導入してもよいでしょう。



2.4 成果物の複製表示

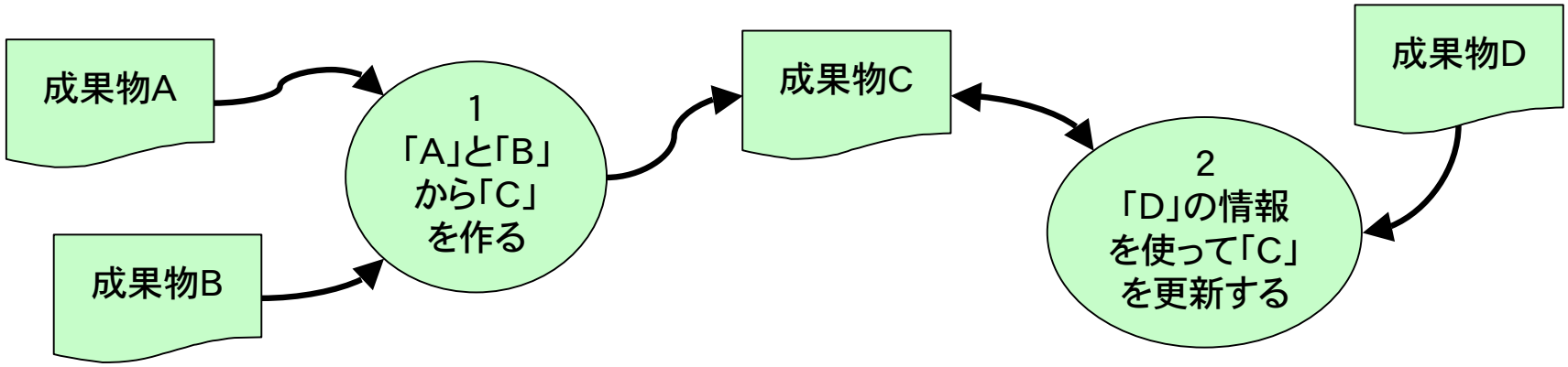
- 一つのPFDの場面の中で、ある成果物が複数のプロセスに関係している場合でも、できるだけフローを伸ばして記述します。
- ただし、関係箇所の距離が離れている場合や、他のフローと交差してPFDの可読性が悪くなる場合は、成果物名または番号の横に「＊」を付ける「複製表示」の方法で対応してください。



  - 「複製表示」は、一つのPFD上に同一の成果物が他にも表示されていることを示します。
  - PFDの層が異なれば複製表示の必要はありません。

2.5 成果物とプロセスをフローでつなぐ

- 「フロー」によって成果物とプロセスをつなぎます。



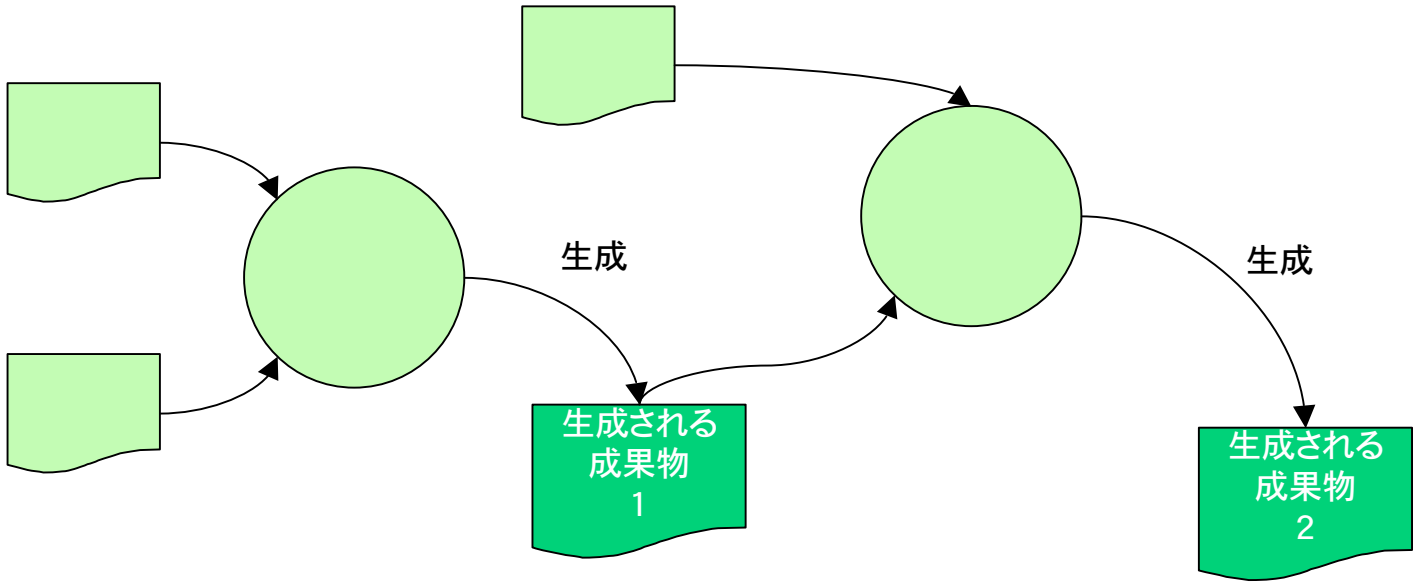
- 成果物「C」が、どの成果物から作られているか
- 成果物「C」が、どのプロセスで活用されているか

が一目でわかります。

- 但し、生成や参照の詳しいことは「プロセス定義」で記述します

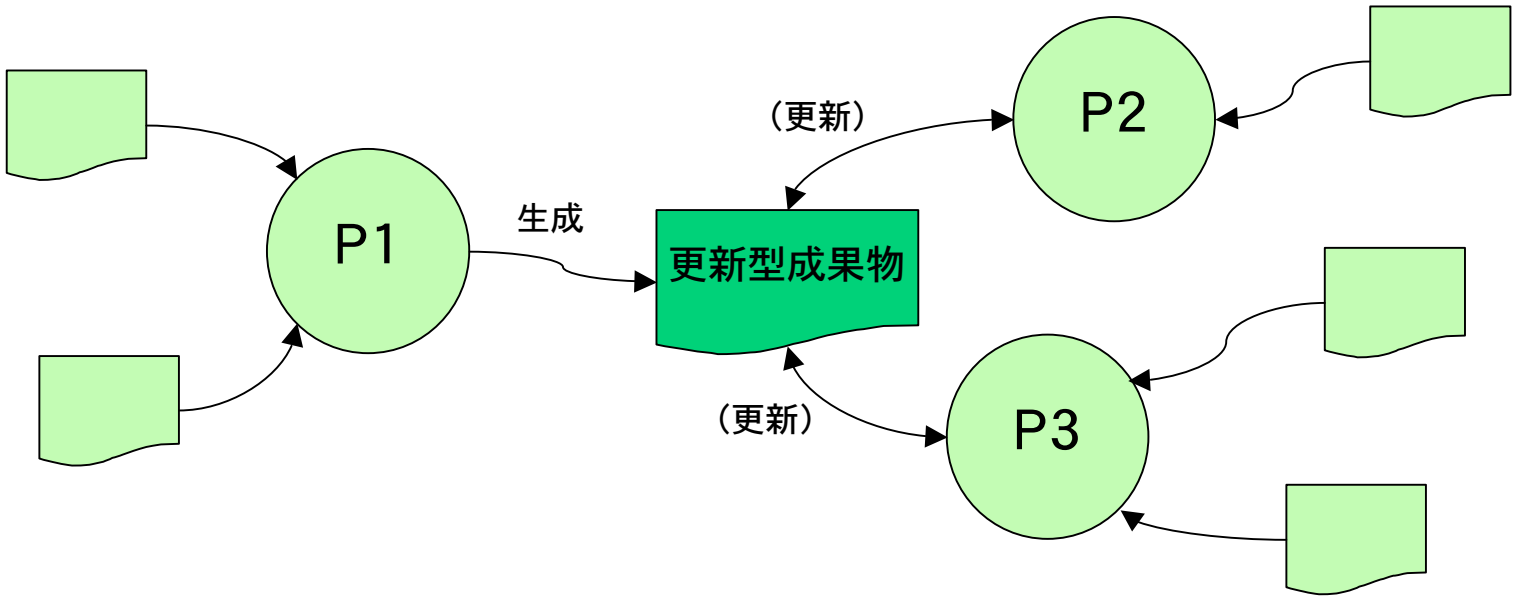
2.6 生成型と更新型 (1)

- 生成型成果物とは、あるプロセスから新規に生み出される成果物のことです。
  - 生成された成果物は、通常は以降のプロセスの入力に使われます。
  - 下図の場合、「生成される成果物1」と「生成される成果物2」は、この一連の作業の結果として、ともに残される成果物であることを示しています。



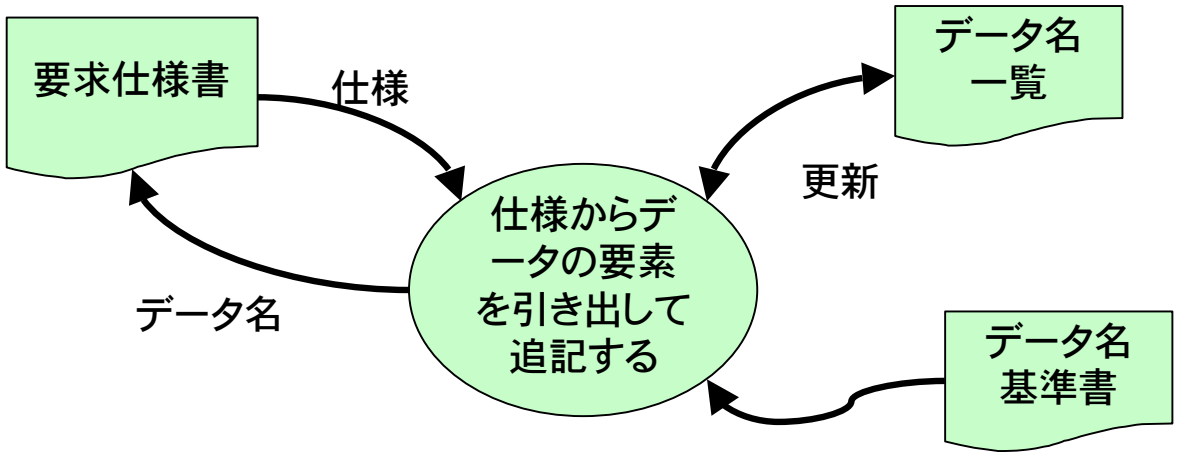
2.6 生成型と更新型 (2)

- 更新型成果物とは、一度「生成」された成果物が、その後に他のプロセスから内容が「追記」されたり、「更新」されたりする成果物のことです。
  - 下図の場合、「P1」によって生成されたあと、「P2」や「P3」によって更新されることを示しています。
  - 更新型を使うことによって、順序性の表現はPFDから消えます。
  - 大事なことは、実際に生み出さない成果物は表現しないようにすることです。



2.7 往復フローと2本の単方向フローの使い分け

- 成果物があるプロセスによって更新される場合、通常は「更新型」のフローを使って表現します。
  - プロセス名などで「更新」される内容が想像できる場合があります。
- 成果物から読み出す情報や書き出す内容をフロー上で見せたい場合は、「更新型」のフローを使わずに入力と出力の二本のフローを使って表現します。
  - この場合は、フロー上に適切な情報を記述すると良いでしょう。

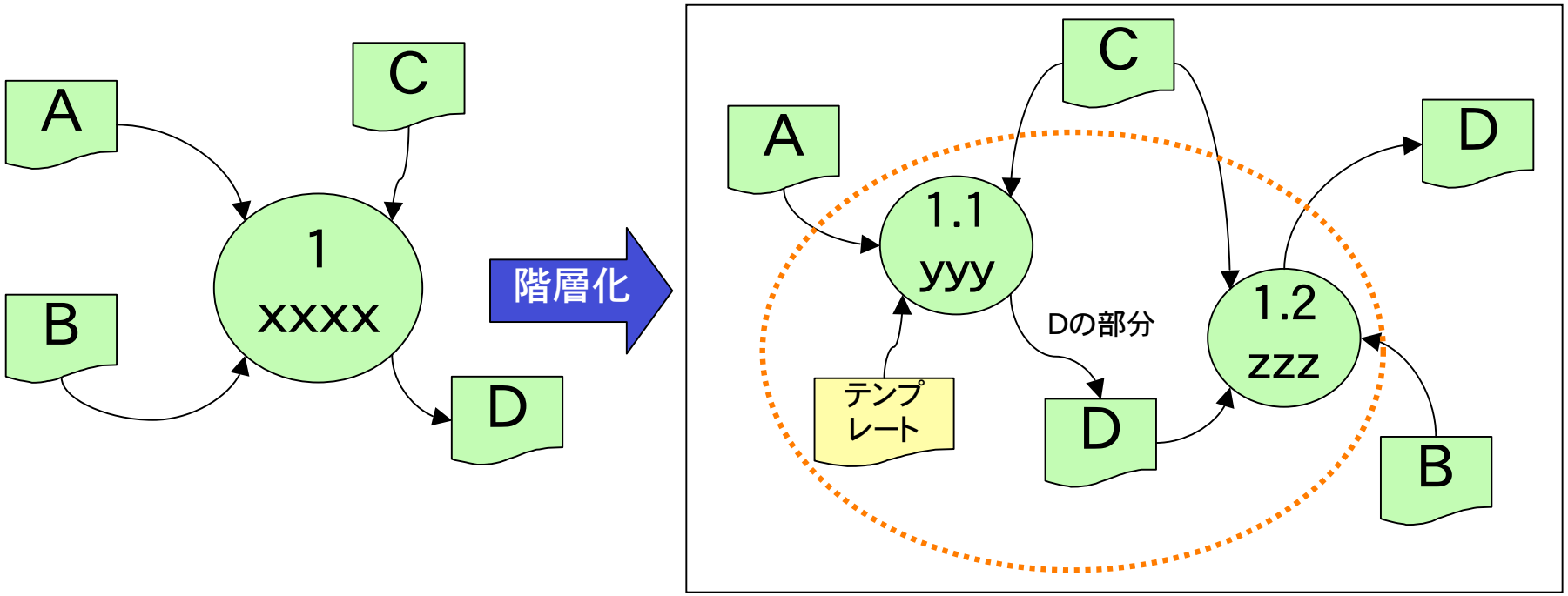


2.8 プロセスの階層化（1）

- 1枚のシートに必要以上に多くのプロセスを配置しないでください。
  - 「7±2」個を目処として、できるだけ階層化しましょう。
  - 下位層に幾つかのプロセスを持っているプロセスであることが一目で分かるように、プロセスの記号が二重線のプロセスに変わります。
  - 逆に、最上位のPFDは、最初から階層化を想定して設計することで、「全体→詳細」という思考パターンが身に付きやすくなります。
- 最上位のPFDにおいて全面的に階層化を活用することで、最上位のPFDは「組織標準」の候補として使うことができます。
  - ただし、このままで「組織標準」にしないでください。
- 階層の上下間では、「親子間のバランス」を確保してください。
  - 上位のプロセスに接する成果物は、下位層のPFDの生成関係と一致するようにしてください。
  - 「テンプレート」などの成果物は下位層のPFDだけに現れてもよいでしょう。

2.8プロセスの階層化（2）

- 上位プロセスに入力／出力する成果物は、下位のPFDに於ても、その関係を維持してください。
  - 「テンプレート」や「ガイドライン」の類いは例外とします。
- プロセスの「番号」によって階層の様子（深さ）が見えます。



2.9 プロセスに番号を付ける

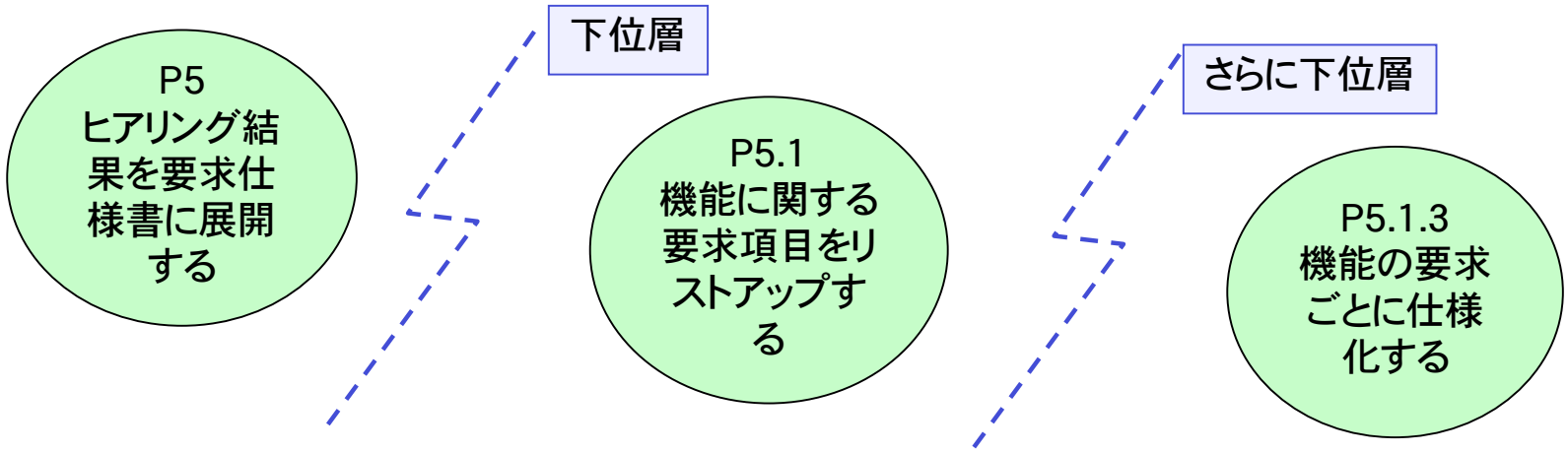
- プロセスには「番号」を付けます。
  - プロセス番号は単なる識別子であって、実行の順序を示すものではありませんが、素直に付けてください。
  - 番号が飛んでもかまいません。
  - また番号は「階層」を表現します。

「P5　ヒアリング結果を要求仕様書に展開する」

「P5.1　機能に関する要求項目をリストアップする」

番号から、このプロセスの「親プロセス」がすぐにわかります。

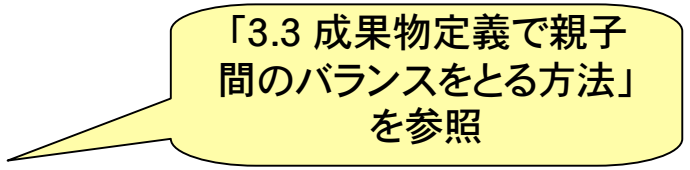
“P”はプロセスの番号であることを示します。



2.10 成果物にも番号を付ける

- 成果物にも「番号」を付けることをお勧めします。
  - 生成する成果物はすべて最上位層に現れています。
    「D10 要求仕様書」

  - 成果物の番号もPFDの「階層」関係の中で使用するときに役に立ちます。例えば、下位層では「D10 要求仕様書」の一部分を生成するプロセスに展開されることがあり、その場合は、
    「D10.2 ○○機能の要求仕様書」
    のように記述することができます。



  - ただしこれができるのは、成果物定義書において当該成果物の構成がきちんと定義されていることが前提です。
  - でもこれによって、この文書は「D10」の一部であることがわかります。

2.11 成果物の定義

- 成果物に対しても「定義書」が必要です。
  - 成果物は最上位のPFDに現れますので、その情報を参考にして成果物定義書を作成します。
  - 内容
    - この成果物の目的や編集方針
    - 成果物の構成と各構成に記述される内容
  - 構成の記述は目次のような記述でも構いませんし、次ページのサンプルのような記述でも構いません。
- 成果物定義があることで、これを入力したり出力するプロセスの定義の信憑性を裏付けることになります。
  - 成果物の定義が曖昧だと、プロセスの定義が書けません。
- 最初にある程度の定義をしてからプロセスの定義を行います。プロセスを定義していく中で、成果物の定義が不十分な部分が見えてきますので、その時点で改めて成果物の定義を修正します。

2.12 成果物定義のサンプル

- 定義書はExcel を使用します。
  - レイアウトは以下のフォームをお勧めします。

文書番号			成 果 物 名 ・ ファイル名	作成日
Ｄ１０			変更依頼書	2007.1.30
	成果物の目的		変更の依頼内容を記述したもの	
	編集方針		依頼者が希望する変更したい仕様の内容や追加機能の概要を箇条書きにしたもの	
			事前に何度か打ち合わせをしているが｛USDM」の形式にはなっていない	
	成果物の構成と簡単内容			
		1	機能追加の要求	
			＝｛追加して欲しい機能名	
			＋当該機能の動作概略	
			＋１｛特に実現して欲しい動作および仕様｝n｝	
		2	変更の要求	
			＝｛変更対象の機能名	
			／＊変更がある仕様がある機能名。ベースの機能仕様書の構成順に並べる＊／	
			＋1｛変更して欲しい項目｜	
			／＊機能仕様書に書かれている仕様に対する変更＊／	
			＋　削除して欲しい項目｜	
			／＊当該機能から削除して欲しい仕様＊／	
			＋　追加して欲しい項目｝n｝	
			／＊当該機能に付かして欲しい仕様＊／	

構成表す番号。成果物の「枝番号」はこの番号を使用します

2.13 成果物の構成の定義方法

- 成果物定義の中の「構成」をきちんと定義しないと、プロセス定義が行き詰まります。
- 定義方法は以下のいずれでもかまいません。
  - ① データの構造体や成果物の目次のように表現する方法
    - 慣れない時は、この方法でも良い
  - ② 構造化分析のデータディクショナリの記述ルールを使う方法

定義	f = ～	「f は右辺の～のように定義される」という意味
結合	＋	構成要素の結合をあらわす
繰り返し	a{ }b a{ } { }b	{ }内の項目が繰繰り返される 繰り返し回数は、a回以上、b回以下 回数表現を省略したときは「0」回以上
選択	[a | b]	[ ]内の項目から1つを選択する
オプション	(a)	( )内の項目は省略してもよい
コメント	／＊　＊／	／*と*／の間にコメントを記述する

2.14 プロセスの定義

- 当然ですが、プロセスにも「定義書」が必要です。
  - PFDの「図（絵?）」を書いただけでは、定義したことにはなりません。
- プロセスの定義
  - 原則として、下位層を持たないプロセスに定義書を付けます。
    - 自明のプロセスや、組織にとって問題のないプロセスは省いても良いでしょう。
  - 内容
    - 実行条件
    - 実際に行う作業内容など
    - 箇条書きでも、フローチャートでもよい
  - プロセスの定義は、入力する成果物から、出力する成果物が生成するアルゴリズムを具体的に記述します。
  - シミュレーションの精度を確保するには、丁寧な記述が求められます。
  - 下位層を持つプロセスの定義書を記述する場合は、下位層のプロセスの実行順序や、相互の関連などを書きます（省略可能です）。

2.15 プロセス定義のサンプル

- プロセスの定義書もExcel で定義します。
  - レイアウトは以下のフォーマットをお勧めします。

プロセス定義書

プロセス番号			プロセス名	作成日
P１．１			ベースの設計書を調べて調査項目（機能）をリストアップする	
	入力情報		D１０ D３	変更依頼書 ベースのタスク設計書
	出力情報		D11	調査資料
	作業内容			作業担当者
	プロセス実施条件：		変更依頼書（D１０）が届いた時点 ベースのタスク設計書などは事前に確保しておく	
	必要スキル：			
	担当者：			
	作業内容：	1	今回の変更依頼【D１０】の内容から、関係する機能およびサブ機能を調査資料【D１１】にリストアップする	
		2	タスク設計書【D３】から該当箇所を探し、参考になる箇所を調査資料【D１１】に追記する この時、資料として不十分なことが判明したときは調査資料【D１１】の調査範囲に追記する	
		3	調査範囲に記述された項目から、調査に必要な工数を見積もって調査資料【D１１】に追記する	
	プロセス終了判定：		変更依頼の内容に対して、関連資料が把握できたり、調査が必要な項目が拾い出された状態	

2.16 関係を表すだけ

- PFDは、成果物とプロセスの“関係”を表すだけで、“順序”を表現しません。順序を表現すれば「PFD」ではなくなります。
  - PFDは“ワークフロー”ではありません。
  - その成果物が
    - ① どのプロセスから生み出され
    - ② どのプロセスで活用されるか

    を表現するだけです。
  - 一つの成果物は、一つのプロセスから作り出されるとは限りません。
    - 最初に1つのプロセスで生成された後は、いくつかのプロセスによって内容が追加されたり更新されることがあります。
    - その様子を正確に表現することが大事です。
  - 一つの成果物は、そのあと複数のプロセスの入力に使われることがあります。
    - それぞれのプロセスで、この成果物のどの部分を使うのかを示すことが大事です。

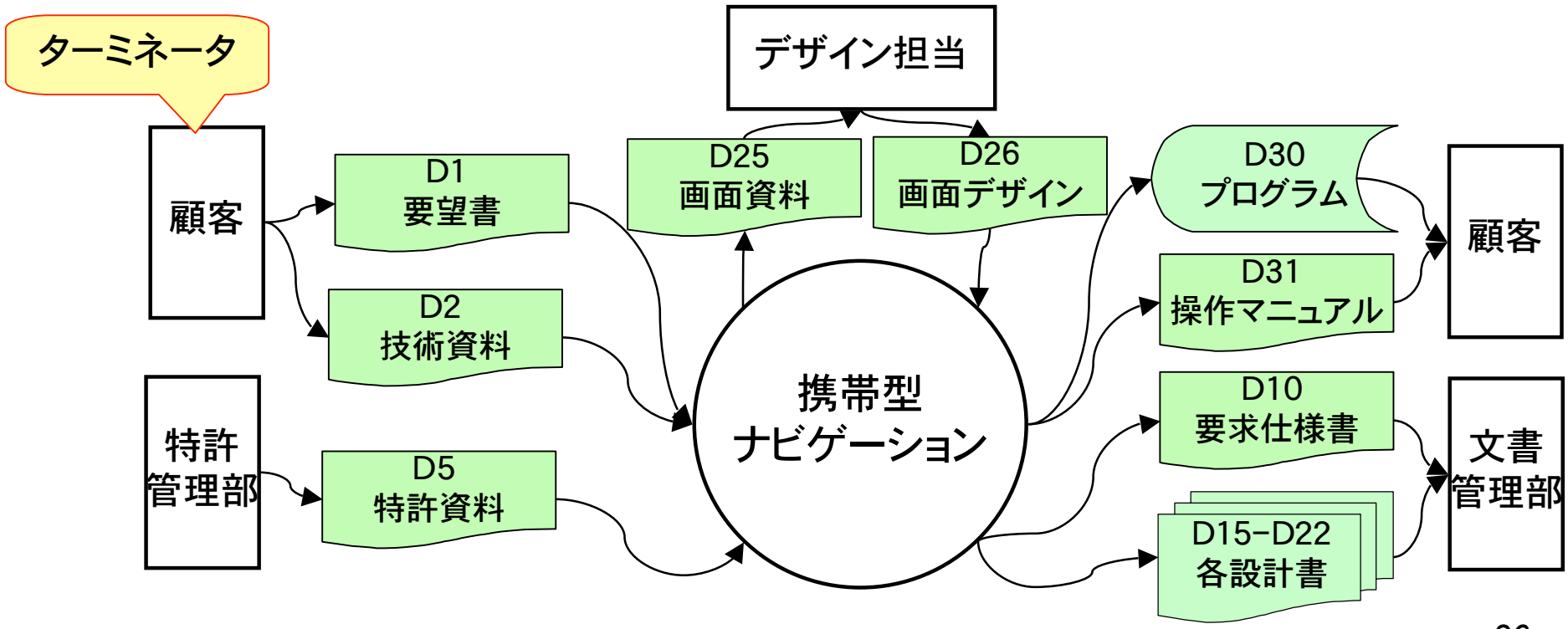
3. PFDの書き方（応用）

- ここでは、PFDを書くときの注意などを簡単に説明します。

1. コンテキストダイアグラムから入る方法
2. 下位層にいきなり現れてもよい成果物とは
3. 成果物定義で親子間のバランスを取る方法
4. 成果物のグループ表示
5. 更新型の有効性
6. 定型パターンの扱い
7. フロー上のデータの表現
8. 無形成果物の表現
9. コメント記号
10. トリガー記号の使い方
11. 「調査する」プロセスの表現
12. サイクルパターンの扱い
13. 使用しないパターン

3.1 コンテキストダイアグラムから入る方法

- 「PFD−0」が最上位のPFDですが、その前にシステムの範囲（境界）を明らかにしたいときは「コンテキストダイアグラム（CD）」を作ります。
  - 周囲に「ターミネータ」を配置し、システムとの間の成果物を明確にします。
    - 最初に成果物を決めておくことで、ムダな成果物を作らないようにします。
  - 「人」「部門」「装置」「棚」「システム」などが「ターミネータ」になります。

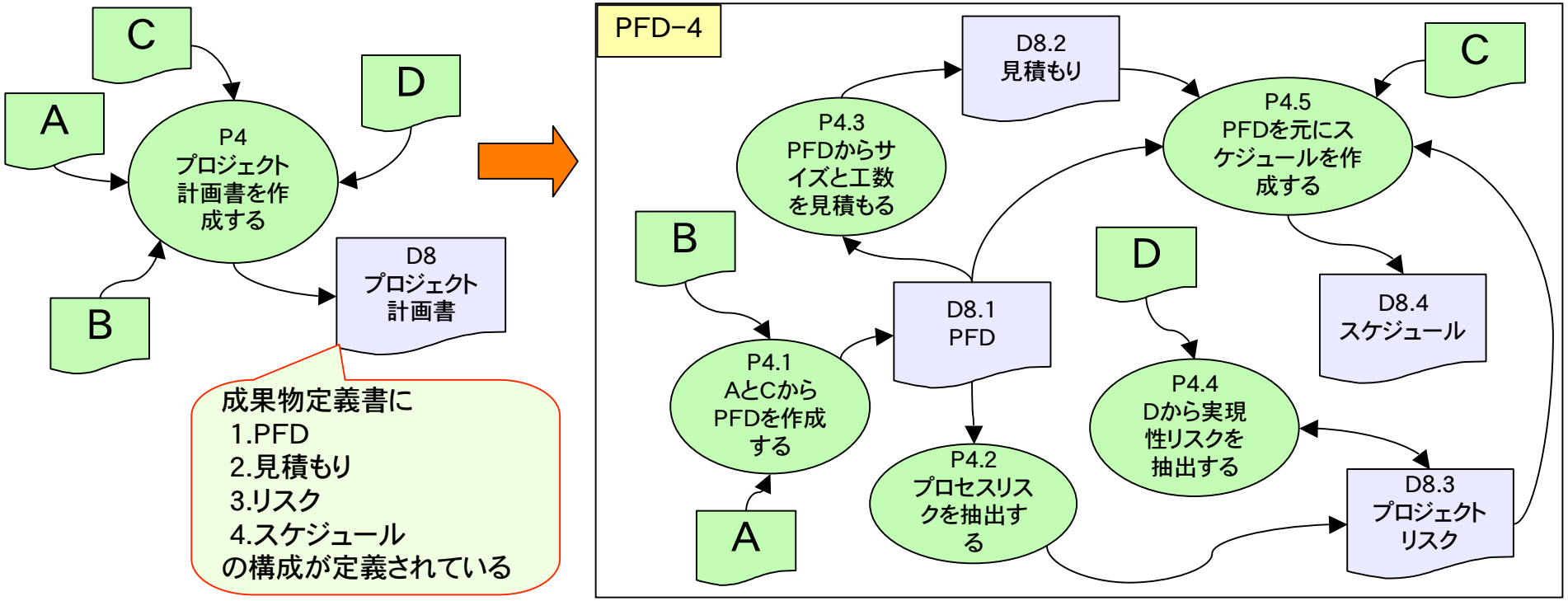


3.2 下位層に直接現れてもよい成果物とは

- すべての成果物は最上位のPFD（PFD-0）に現れるというPFDの基本ルールによって、プロセスの親子間の整合性を確保しています。
- しかしながら、成果物の中には同じ成果物があちこちのPFDに現れる割には親子間の整合性をあまり損ねないものもあります。
  - レビュー議事録
  - レビュー指摘リスト
  - プロセスの手順書やガイドライン、記述規約 など
- これらの成果物に対しては、親子間のバランスを無視する形で、それを使用するPFD上に直接表現することを認めています。
  - ただし、成果物の種類には明確な「歯止め」をかけておいてください。

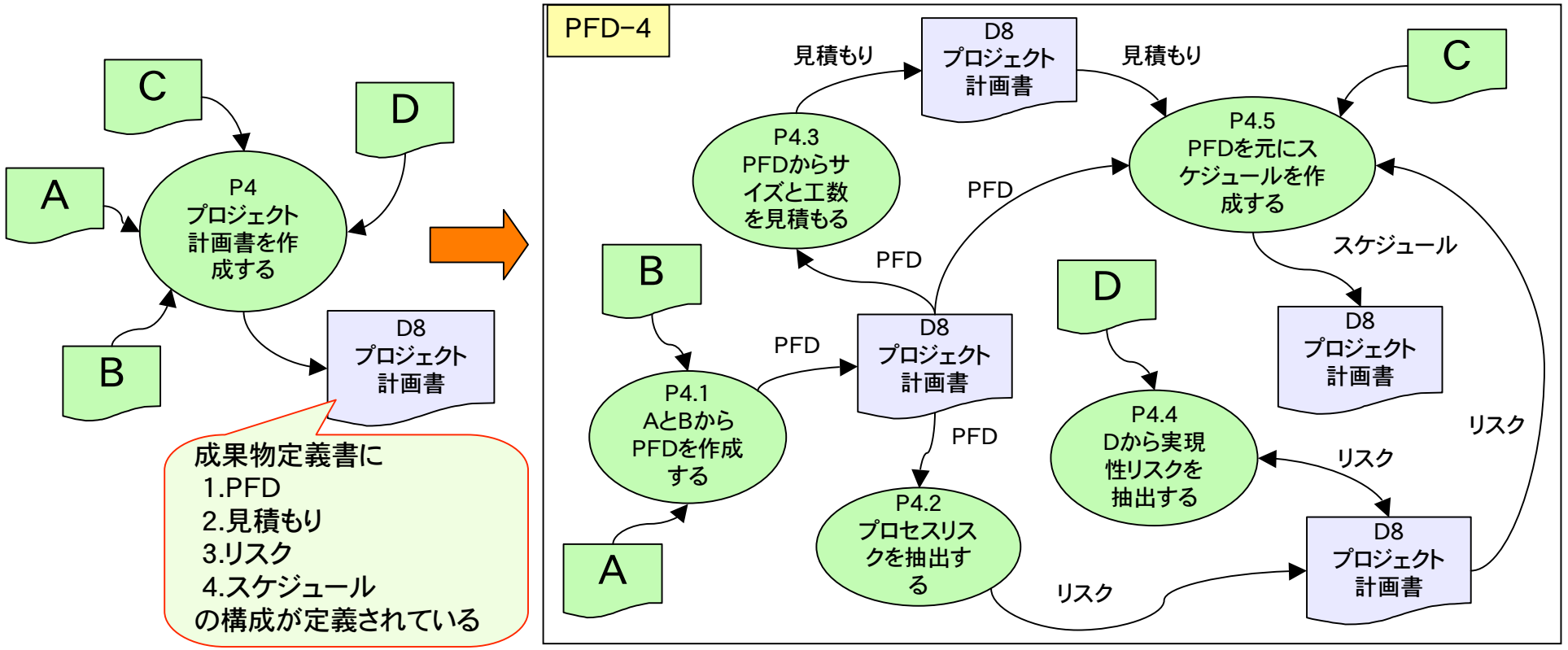
3.3 成果物定義で親子間のバランスを取る方法(1)

- 上位層のプロセスと下位層のPFD間の「親子間のバランス」の取り方として、成果物の「枝番号」を使う方法があります。
  - 下例では、「D8.1」～「D8.4」を生成することで「D8」を生成したことと同じ意味になり、「プロジェクト計画書に統合する」プロセスは不要になります。



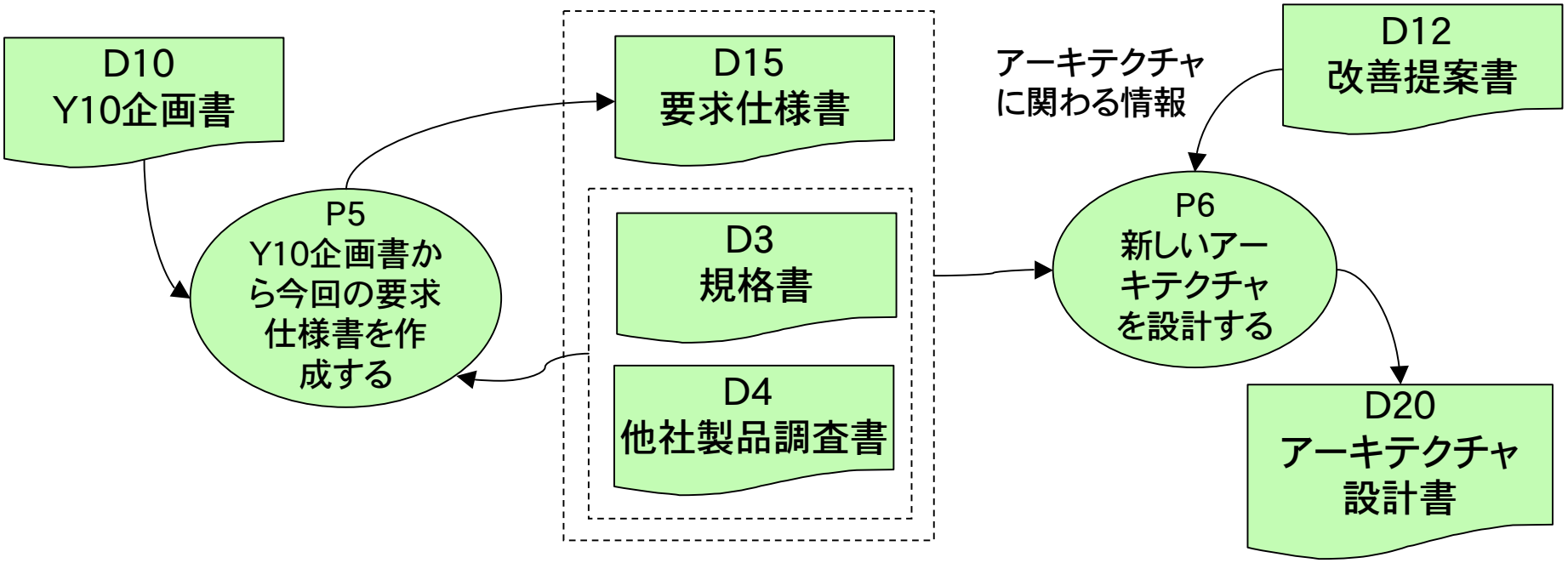
3.3 成果物定義で親子間のバランスを取る方法(2)

- 成果物の「枝番号」の代わりに、「フロー情報」で「親子間のバランス」を取ることもできます。
  - この場合でも成果物定義で成果物の構成を定義しておいてください。構成の定義がなければ、フロー情報の裏付けを失います。



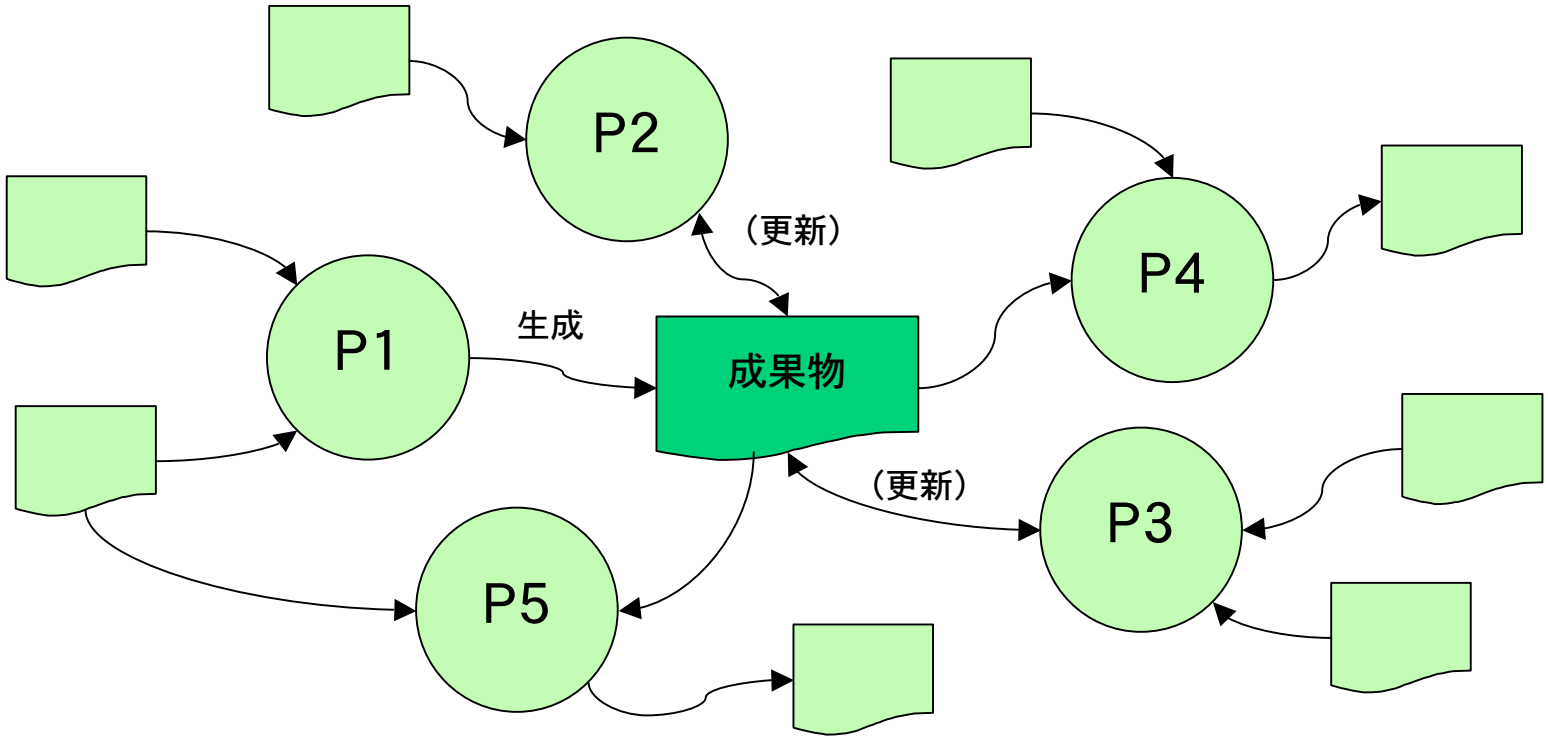
3.4 成果物のグループ表示

- 一つのプロセスに対してもたくさんの成果物が関与し、成果物とプロセスを結ぶフローが多くなってPFDの可読性を損ねるときは、成果物を点線の枠で囲む「グループ表示」の方法を使ってください。
  - 成果物の配置とグループ表示を使うことで、ずいぶん整理されます。



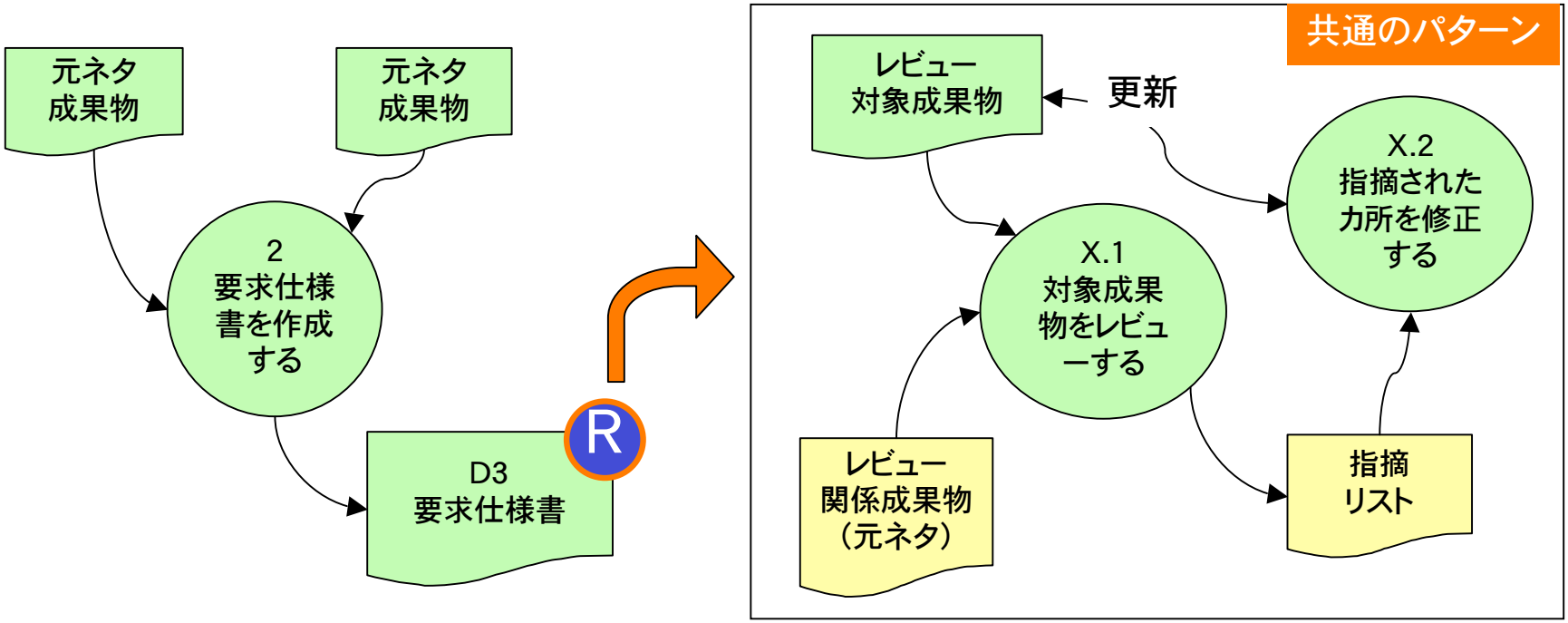
3.5 更新型の有効性

- 「P4」「P5」などの以降のプロセスにおいてこの成果物がどのように使われるかによって、「P2」「P3」の順序性が決まることがあります。
- またこの性質を利用して、上位（「P1」以前）の作業の遅れを「P2」「P3」での作業の順序を工夫することで、隠せることがあります。



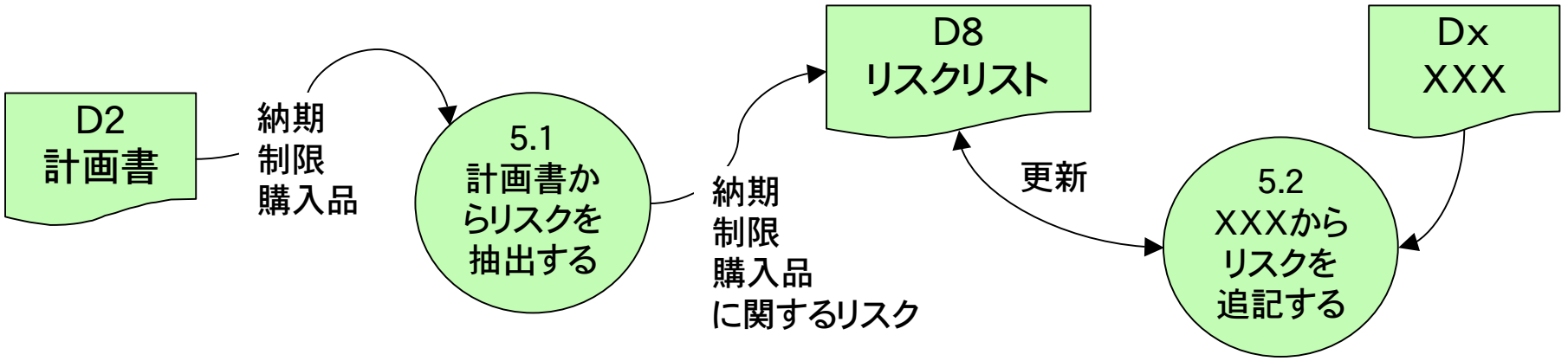
3.6 定型パターンの扱い

- レビューのようなプロセスをPFD上の全ての成果物に対して表現すると、PFDの可読性が悪くなりますので特殊な記号を使用します。
  - 組織の中でパターンを共有し、PFD上では「記号」で表現するとよいでしょう。
  - ただし、レビュープロセスでも、そのプロセスを強調したり、レビューでの成果物の活用を表現するために意図的にプロセスとして表現することがあります。



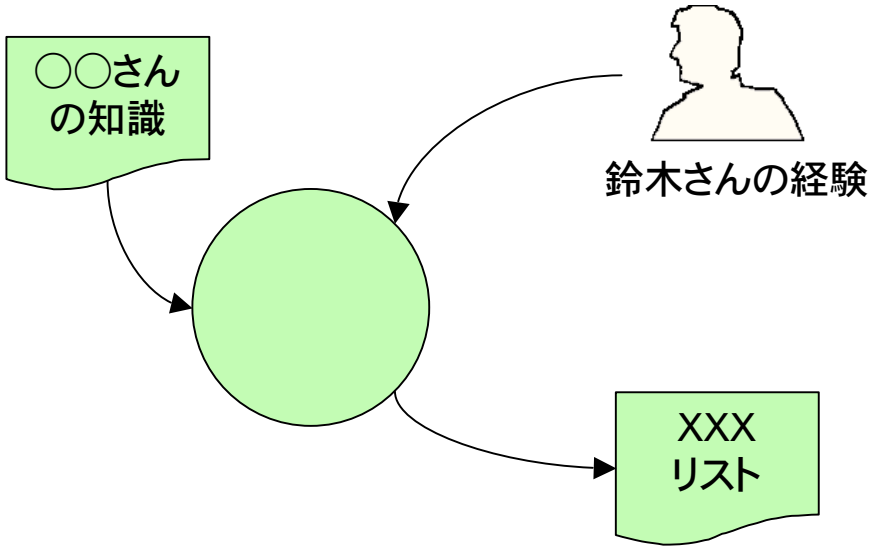
3.7 フロー上のデータの表現

- フロー上には、必要に応じて適切に情報を付加することができます。
  - 入力する成果物のすべてを使用するのではない場合、そのプロセスに関係する成果物の要素を表現します。
    - 更新型の場合、「更新」と表現する代わりに、そこで追記される項目を表現してもよいでしょう。
  - こによってシミュレーションをスムースにする効果が得られます。
  - ただし、これを多用するとPFDが見にくくなりますので注意が必要です。



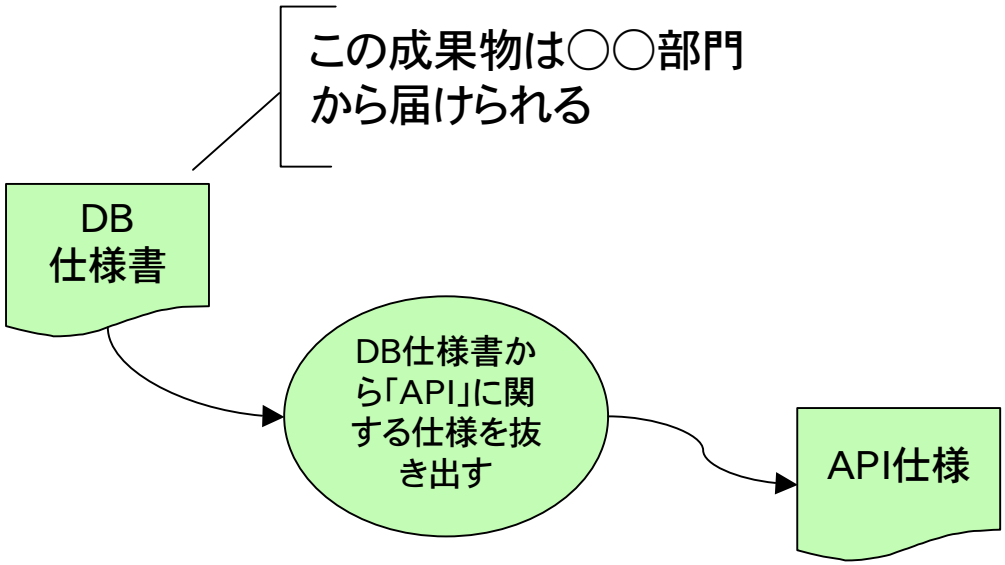
3.8 無形成果物の表現

- 実際の作業では、入力源として、必ずしも「形」になった成果物が存在するとは限りません。
  - 特に、上流のプロセスでは、経験豊かな人の「ノウハウ」や「知識」を活用するプロセスも存在しますので、上手に表現してください。
    - 成果物の図を使っても構いませんし、特別な図を使っても構いません。
  - ただし、「鈴木さんの経験」も「成果物定義」が必要で、そこでどのような経験内容が活用されるのかを記述します。



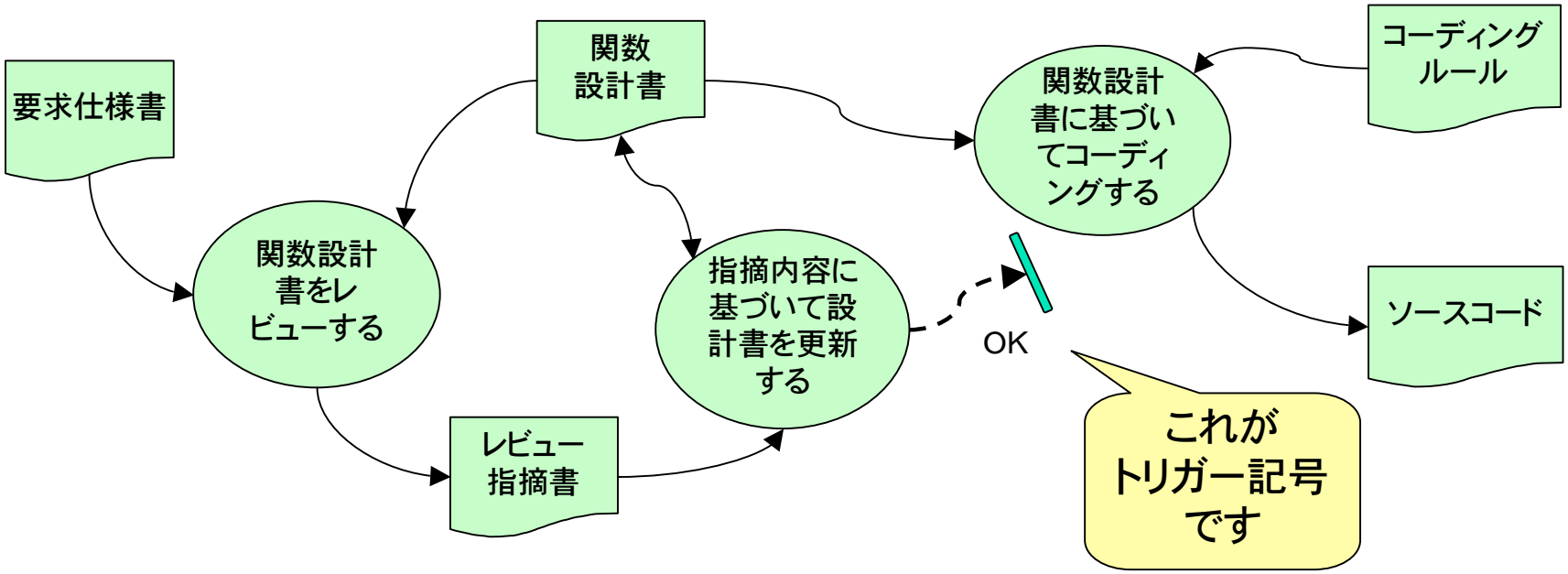
3.9 コメントの書き方

- PFD上の成果物やプロセスに注釈を付けておきたい時は、注釈の記号を使って、目的の成果物やプロセスの近くに配置してください。
  - ただし、多用するとPFDの可読性を損ねますので、注意が必要です。



3.10 トリガー記号の使い方

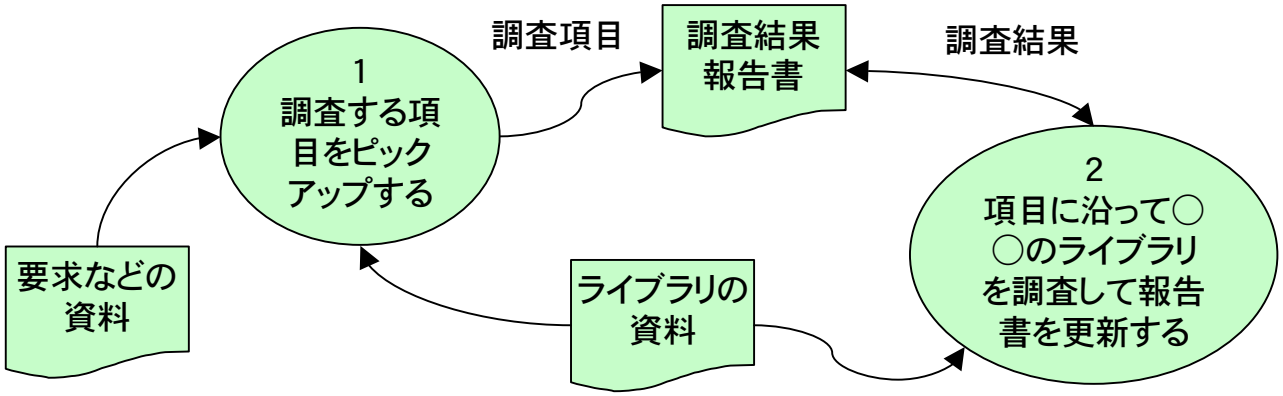
- PFDでは、原則としてプロセスの順序を表現しません。
- あるプロセスの結果から、次に起動されるプロセスをどうしても表現したいときは「トリガー記号」を使って、その「順序」を表します。
  - 通常は、成果物が渡されることで起動しますので、必要ありません。
  - また、図を見れば合理的な順序はイメージできますので、多用を避けることをお勧めします。



3.11「調査する」プロセスの表現

- 「・・・を調査する」というプロセスは一般に以下のように表現されます。
  - しかしながら以下のような表現では、「調査する」というプロセスの工数が見積れなくなります。

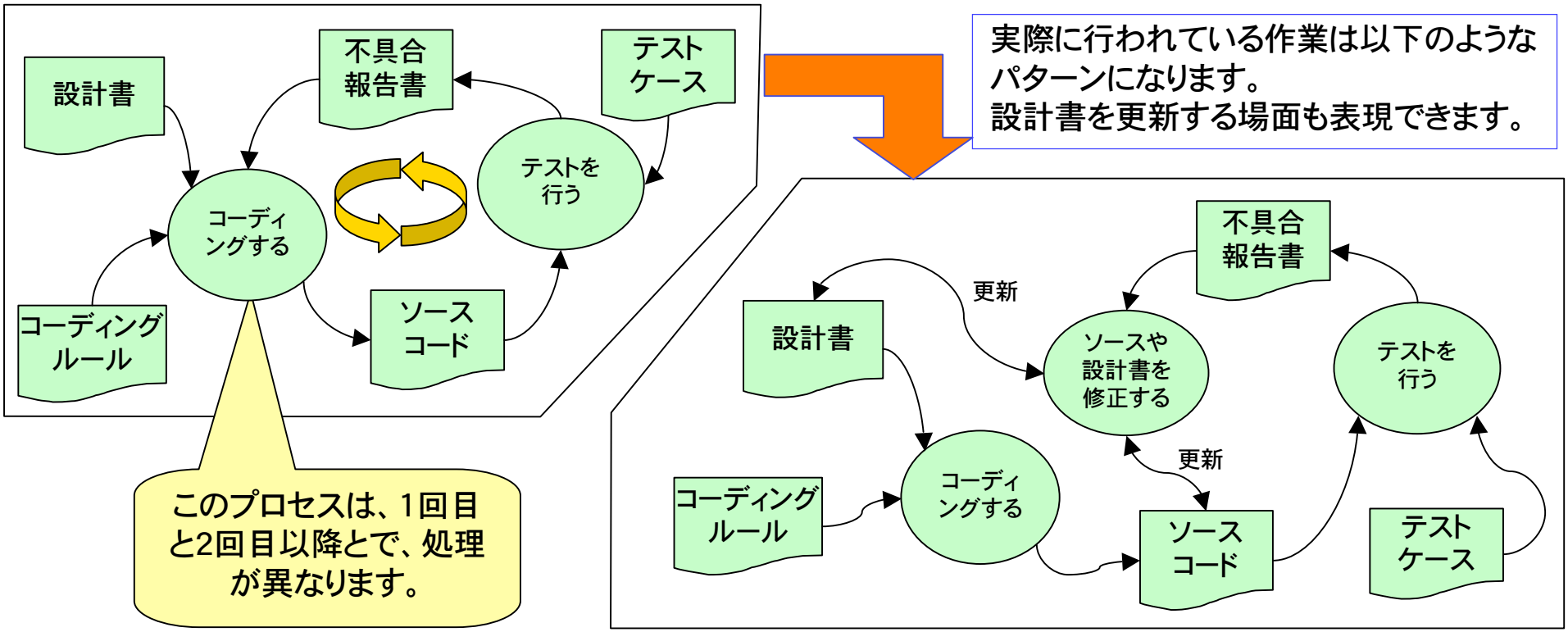
- 調査項目をピックアップするプロセスと調査するプロセスを分けることで、見積りの問題もクリアできます。



- 調査項目数の見積りに基づいて「2」のプロセスの工数を見積ることができます。
- 「1」のプロセスの工数の見積りは小さいので、誤差がでても大きな問題にならない。
- 「1」のプロセスの実績値によって項目数と項目の内容が見えるので、「2」のプロセスの工数を調整できる。

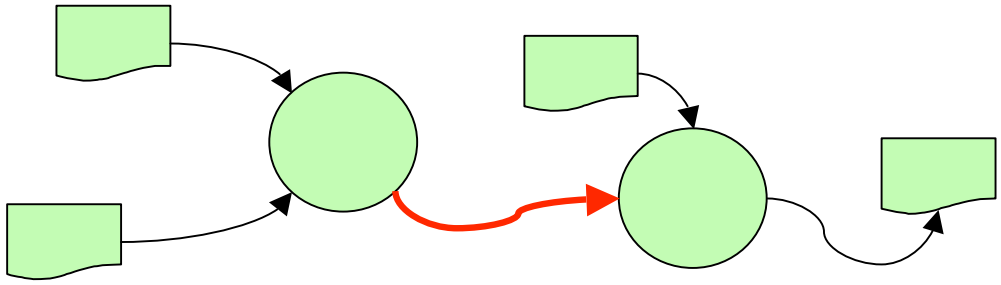
3.12 サイクルパターンの扱い

- 幾つかのプロセスが繋がって“ぐるぐる回る”「サイクルパターン」が形成されるときは注意が必要です。
  - 1つのプロセスに次元の異なる複数の作業が混在する可能性があります。
  - モジュールの尺度の「論理的凝集」と通じるところがありますので、回避してください。

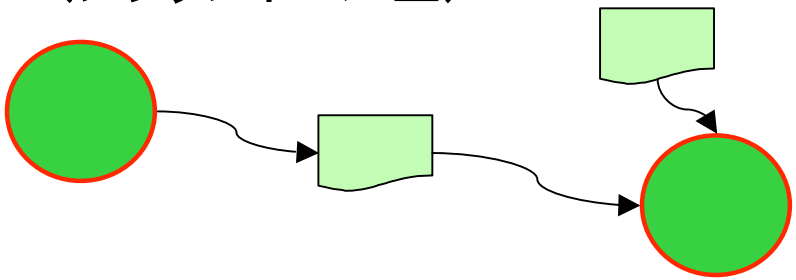


3.13 使用しないパターン

- PFDの表記法では、以下のパターンは避けるようにしています。
  - プロセスとプロセスが繋がっている状態
    （玉突き型）



  - 入力の成果物を持たないプロセス
    （ビッグバン型）
  - 出力の成果物を持たないプロセス
    （ブラックホール型）



- 作業の順序をイメージしていることが多く、受け渡している成果物を探します。
- 成果物が渡されていないときは「トリガー」を使います。

- 他のグループとの受け渡しをイメージしているのであれば、成果物で受け渡すようにします。
- 成果物のところに必要なコメントでも書いてください。

4. PFDの効果と活用

- ここでは、PFDを書くことの効果や活用方法について説明します。

1. 成果物一覧が把握出来る
2. 成果物の連鎖を使った見積もりの仕方
3. 簡単にスケジューラに展開できる
4. 進捗管理の重要なツール
5. バグの原因プロセスの追跡が容易
6. PPQA(SQA)の活動のネタが見える
7. 複数のチームの作業を扱う
8. 別案を考え出すツール
9. プロセスを設計するスキルを失わない

4.1 成果物一覧が把握出来る

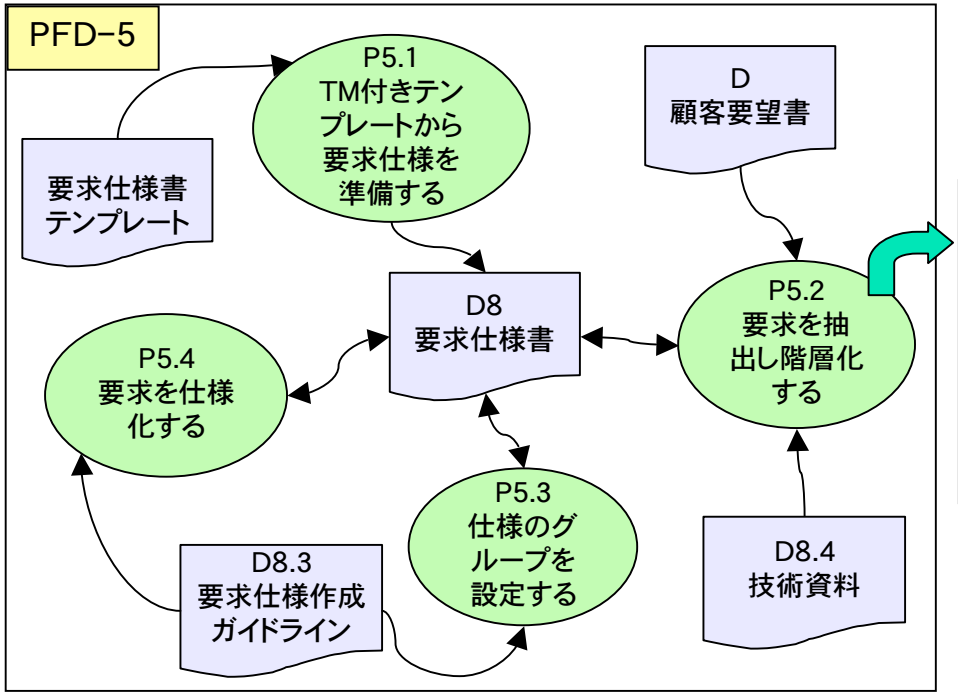
- プロジェクトの計画書の中の「成果物一覧」はいつも同じパターンになっていませんか?
  - 確かに、毎回大きく変わることはないのですが、開発アプローチの前半はいつも同じとは限りません。
  - 今回の要求を仕様化するのに、何を調べなければならないのか?
  - 今回は他社製品との比較調査したものが必要ではないのか?
- PFDは、今回の要求を満たすための合理的な開発アプローチを設計し、何度もシミュレーションしています。
- その結果、最上位のPFD(PFD-0)に、参照する資料や作成する成果物がすべて見えますので、ここから「成果物一覧」を作ることができます。
  - 作成する成果物の構成も把握出来ています。
  - 「構成管理」の対象とする成果物もここで指定でき、「成果物一覧」を使って「版管理」が可能になります。

4.2 成果物の連鎖を使って見積もりができる

- 見積もりの原点は「サイズ」です。
- ここでは事前に「成果物一覧」が把握されていますので、この一覧に「見積もり」や「サイズ」の欄を設けてそこに見積もり値を記入します。
  - 成果物のサイズが見積られれば、PFDからそれを作ることに関与するプロセスを特定し、その生産性を仮定し、さらにそこから工数を算出します。
- PFDから、成果物はその「尺度」によって連鎖していることが見えます。
  - 要求の項目数が仕様の項目数に繋がります。
  - 仕様の項目数が関数の数につがり、さらにソースコードの行数に繋がります。
- ただし、成果物によってはサイズを計るための「尺度」を複数持っていて、成果物単位で一つの「サイズ」を見積ることができない場合は、PFDにある「成果物定義書」を使います。
  - 成果物定義書の「構成」の単位で「尺度」を設定し「サイズ」を見積もります。
  - この成果物に関与するプロセスの定義書から、成果物の「構成」単位の作業項目を探し、そこで仮定される生産性から工数を算出します。
  - 成果物一覧には、これらの項目毎の工数を合計したものを使います。

4.3 PFDからスケジュールに容易に展開できる

- 多くの人は、スケジューラを前にして「次は何をしようか」と迷っています。
- でもPFDとプロセス定義書を使えばスケジュールに簡単に展開できます。
  - PFDの機能的合理性の検証も終えていて作業の順序もほとんど見えています。
  - プロセス定義書には具体的な作業内容が記述されていますので、PFDのプロセス名とプロセス定義の「作業内容」の記述からスケジュールに展開します。
  - この時、プロセス定義書の「作業内容」の「C列」に番号が付いた項目を対象にします。



スケジュール

5	要求仕様書を作成する		
	5.1	TM付きテンプレートから要求仕様を準備・・	
	5.2	要求を抽出し階層化する	
		5.2.1	機能名を設定しカテゴリ欄に記入する
		5.2.2	理由をつけて振る舞いを上位要求に・・・
		5.2.3	この段階でレビューして範囲を確認する
		5.2.4	範囲の広い要求を階層化する
	5.3	仕様のグループを設定する	
	5.4	要求を仕様化する	

プロセス定義
P5.2 要求・・

作業内容
1.・・・・
2.・・・・
3.・・・・
4.・・・・

4.4 成果物の連鎖を使った進捗管理ができる

- 進捗管理の原則は、事前に見積った「サイズ」と「生産性」の値の実績値との「差」からこの先の作業を調整することです。
- 見積もりのときに使った、「尺度」による成果物の連鎖を使ってこの先の成果物のサイズ見積もりの値を調整していきます（フィードフォワード）。
  - 成果物一覧のサイズ欄が空欄の場合はPFDの成果物定義の「構成」毎にサイズが見積られていますので、その構成単位の「尺度」で次の成果物に連鎖しているはずです。
  - ある成果物のサイズの実績値が「10%」増えた場合、連鎖しているこの先の成果物のサイズも「10%」増える可能性があり、連動して工数も調整します。
  - その結果、早い段階で開発アプローチの「別案」を考えたり、2ヶ月先のリソースの対応を今から準備したりできるのです。
- 成果物の連鎖が使えなければ「フィードフォワード」は使えません。「フィードフォワード」が使えなければ進捗管理は後ろに遅れるだけの対応になります。

4.5 プロジェクトの後半でリソースの投入が可能に

- 一般に、プロジェクトの中盤から後半にかけてリソース（人員）を投入することはタブーとされています。
  - 原因は、直前になってリソースの確保に走るため、確保された人たちに回す作業が準備できていないことにあります。
  - その反省から、最初に多くのリソースを確保したものの、細かく分割された「担当割り」になって、却って作業を混乱させる結果にも繋がっています。
- PFDをベースにしたサイズによる成果物の連鎖を活用することで、見積もり値がプロジェクトの進行に合わせて調整されていれば、リソースの投入計画も早い段階で調整できます。
  - 中盤や後半で投入される人に割り当てる作業も、場合によっては数ヶ月先から準備できますので、彼らが作業するための入力物の対応も可能です。
- きちんと計画されていれば、後半でのリソース投入も混乱しないのです。

4.6 バグの原因プロセスの追跡が容易

- CMMIなどの効果もあって、「バグの原因はプロセスにある」という認識はだいぶ広がったと思われます。ここから「プロセスの改善」に繋がります。
  - この場合の「原因プロセス」は基本的には「実行してきたプロセス」に問題があるのではないか、ということです。
  - ただし「要求仕様作成プロセス」というプロセスは実際に実行したプロセスではありませんので、このレベルで捉えても何の役にも立ちません。
- 原因プロセスに3種類あります。

① 実行の仕方が不十分だったプロセス
② 必要と分かっていたが納期などの圧力で省いてしまったプロセス
③ そのような有効なプロセスがあることに全く気がついていないプロセス

- テストに入ったところでPFDから実行したプロセスの一覧を作り、バグ報告書の「原因プロセスの特定」ではこのプロセス一覧から選びます。
  - 実行してきたプロセス以外のプロセスが関係していると思われることがありますので、工程レベルごとに「その他プロセス」を設けてください。
  - ただし、この方法でも上記の③の原因プロセスは発見できないと思います。

4.7 PPQA(SQA)の活動のネタが見える

- 実際にその作業をする人がPFDを書くことで、実行技術の不安な箇所が見えます。
  - ただし、誰でも見えるわけではありません。
- PPQAの担当者は、プロジェクト計画をチェックしているときに(またはその前に)このPFDを見ますので以下のような視点でチェックします。
  - PFD上のプロセスや成果物の連鎖にぎこちない箇所はないか?
  - その成果物を生み出すプロセスに強引さはないか?
  - プロセス定義の「作業内容」の記述に具体性がなく淡白になっていないか?
- ここで不安なプロセスが見つかれば以下の対応をします。
  - 担当者と話をして不安な状態を解消するためにアドバイスする。
  - 参考になる過去のPFDと定義書のサンプルを見せて気付かせる。
  - リーダーに「プロセス上のリスク」があることを知らせる(リスク管理で対応)。
  - 進捗段階でのPPQAの支援項目として取り組みの経過を見ながら支援する。
- これらはPFDが問題の箇所を浮かび上がらせているのです。

4.8 複数のチームの作業を扱うことができる

- 複数のチームが集まって取り組むような場合、最上位のPFD(PFD-0)は全体の責任を負うリーダーが書きます。
  - PFD-0の現れるプロセスはおそらくすべて下位層を持つプロセスになります。
  - 最上位層またはその下の層あたりで、担当するチーム別に関わるプロセスを分けるようにします。
- プロセスの単位で担当チームが特定できる状態になったところで、それ以降は担当チームのリーダーを中心にして下位層のPFDを展開します。
  - その下位層では、さらに一人ひとりの担当者が自分の担当範囲でPFDを設計するのが良いでしょう。
- 全体の責任を負うリーダーは、こうして各チームごとに設計されてきた下位層のPFDを見て、承認するか再設計を求めるかを判断します。
  - そのためには、彼自身がPFDを使ったプロセスの設計によってプロジェクトを成功に導いた経験を持つ必要があります。

4.9 PFDは「別案」を考え出すためのツール

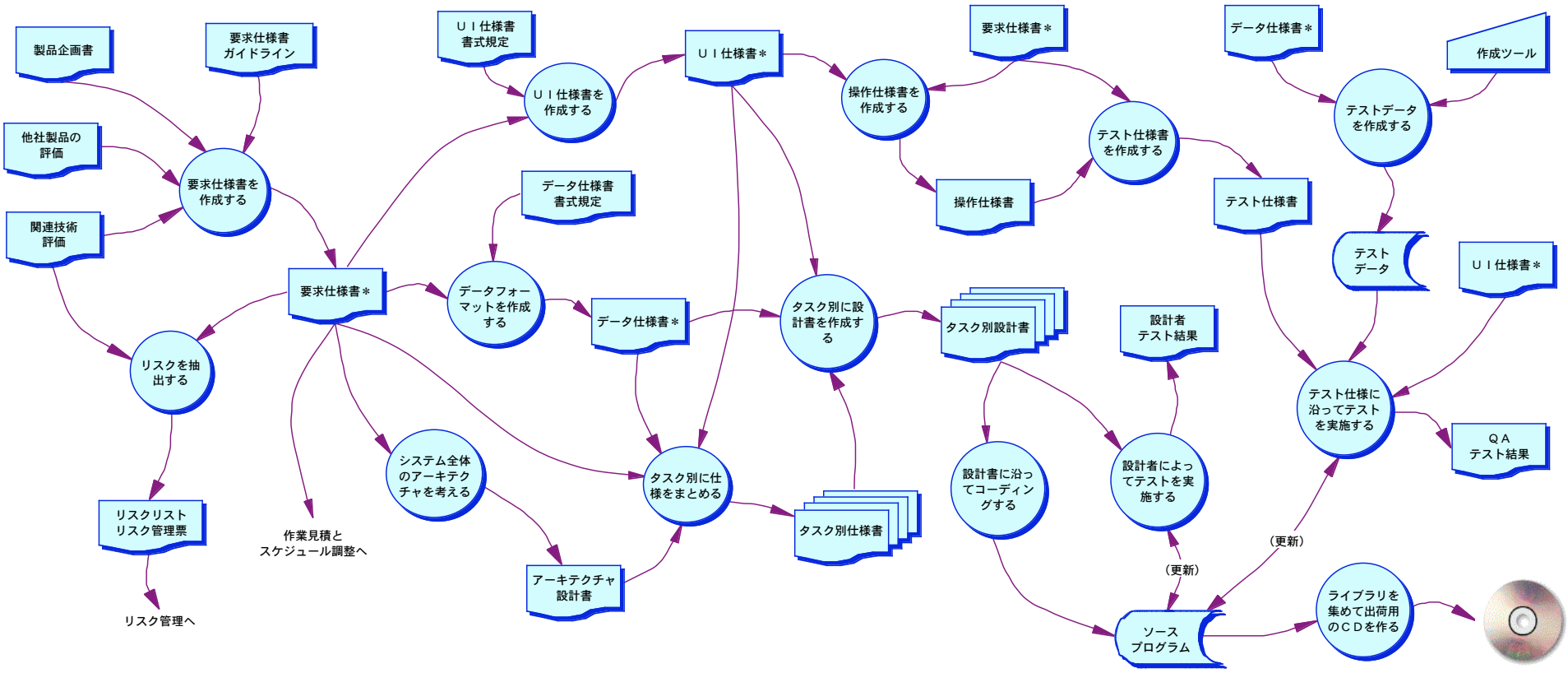
- 計画段階で考えた開発アプローチや、その中の具体的なプロセスのとおりに最後まで進捗することなんてないでしょう。
  - 途中で仕様変更や機能の追加が発生するかもしれません。
  - 最初の見積りの甘さからどんどん予定がずれていくこともあります。
  - ある担当者の作業が思わぬ見込み違いから大きく遅れることもあります。
- このようなときに開発アプローチの「別案」を考え出す必要があります。現実には、これができないためにスケジュールの見直しも後ろに遅れるだけになります。
- PFDは、このようなときに「別案」に気付かせてくれるツールなのです。
  - 必要のないプロセスや成果物はないか?
  - 余計な成果物を作り出そとしていないか?
  - 別の方法で同じ効果の成果物を生み出すことはできないか?
  - プロセス内の作業の順序を変えることで遅れを隠すことはできないか?

4.10 プロセスを設計するスキルを失わないこと

- 大事なのは、実際に作業をする人が自分の責任範囲のプロセスを自在に設計できるスキルを持っていることです。それが組織の強さになります。
  - PSPでも、自分のプロセスを自分で設計し、サイズや工数を自分で見積ることを求めています。
    - PSP：ハンフリー氏が中心となって推進する個人レベルの訓練プログラム
- この能力さえあれば、新しい時代の要求に応え続けることができます。
  - その時に必要なソフトウェアエンジニアリングの技術は提案されているでしょうから、それらの技術を並行して習得することで、プロセスに反映します。
- CMMIでも「テーラリング」という行為によってプロセスを設計するスキルを守ろうとしています。
  - 確かに、毎回「一から」プロセスを設計するのは合理的ではありませんので、選択された「組織標準」からテーラリングするというのは悪くない考えです。
  - でも実際に設計しない人が作ったプロセスや、テーラリングの出番のないそのまま実行できる「組織標準」のプロセスに従って作業をしていれば、変化する時代の要求に応えられない事態に陥ることは明らかです。

おまけ：PFDのサンプル(1)

- 現実には下図のようなイメージになります。
  - 新規開発の最上位層(PFD-0)のパターン



おまけ: PFDのサンプル（2）

- リスク項目を引き出すプロセスのPFD（PFD-0）です。
  - 「更新」を使ってリスクリストが仕上がっていく様子が見えます。