困ってませんか？ 派生開発

～XDDP はじめの一歩～

はじめに：派生開発とは

本団体の名称にもある「派生開発」は、新規開発とは異なる状況での開発を指す言葉です。これには、差分開発、改良保守、改造開発など、さまざまな用語が用いられますが、「派生開発」もそのような用語の一つです。ソフトウェア工学や何らかの規格で明確に定義された用語ではありません。

本書における「派生開発」は、以下のような定義になります。

- ベースとなるシステム（完成品、開発途中を問わず）があり、その部分変更もしくは追加・削除により新しいシステムを開発する
- 既存システムの一部再利用は、派生開発とはしない

このような「派生開発」を行う現場では、短い期間での開発が要求される、ベース・システムを理解しない状態での作業が強いられるなど、さまざまな制限が開発に課せられます。

XDDP(eXtreme Derivative Development Process)は、このような「派生開発」に特化したプロセスを策定しています。

注釈：保守開発と派生開発

ISO／IEC 14764：2006 にてソフトウェアの保守とは、「ソフトウェアの完全性を維持しながら、既存のソフトウェア製品に対して修正を行うこと」と定義されています。「保守開発」という言葉はこのプロセスを示す言葉です。保守開発は、開発したシステムに何らかの「保守」を行う事を意味するのに対し、派生開発は、開発したシステムを元に新たなシステムを開発することを意味しています。

１章：XDDP のポリシー

派生開発では、短納期や低コストという条件が課される状況が多くあります。そのような制約の中で、ベースが大規模で把握しきれない、過去の設計情報が失われた、などの諸事情によってデグレードが発生することがあります。新規開発とは異なった、このような特徴的な問題を認識したうえでなければ、適した対策はできません。

XDDP はそのような派生開発特有の問題点に着目し、派生開発に特化した開発手法ですが、XDDP を理解し、開発現場で適用していくには、学習と経験が必要です。その際に意識すべきことは XDDP が目指していることの理解です。言い換えれば「XDDP のポリシー」を知ることが重要になります。

私たち研究会では XDDP のポリシーを以下のように捉えています。

1. 多角的な視点を使い、コード変更前に問題に気づく
2. 時間経過を利用し、変更仕様の理解を深める
3. レビューを意識した成果物を作成する
4. 上流で問題に向き合い、無駄を排除する

以下は、上記のポリシーについての詳細な説明です。

1. 多角的な視点を使い、コード変更前に問題に気づく

「多角的な視点」を使うとは、仕様変更が与える影響範囲や衝突をさまざまな視点から捉えることを意味します。仕様変更を確認して即座にソースコードを変更してしまうと、他の仕様との関係に気づかずに変更をしてしまうことがありますが、そのようなやり方は手戻りが大きく発生する可能性を高めてしまいます。派生開発は新規開発と異なり、変更が既存の部分へ影響することにより、他の仕様と衝突することが考えられるため、相互の関係を確認していくことが必要です。XDDP では What、Why、Where、How の視点を使うことで他の部分との関係の確認を強化しています。

変更要求仕様書では What：何を変更するか、Why：なぜこの仕様変更が必要なのかの視点で見ます。TM ではシステム全体を俯瞰し、Where：どこに変更箇所があるかを見ます。そして、変更設計書では How：どのように変更するかの視点で詳細を考えます。これらの多角的な視点で見ることで、ミスが入り込む余地を少なくしています。

2. 時間経過を利用し、変更仕様の理解を深める

XDDP は、変更要求仕様書で、何を変更し、なぜそうするかを文書化します。次に TM と変更設計書により、何処をどうやって変更するかを検討した後にソースコードを変更する流れになっています。派生開発では、変更に対して別の仕様との関係が後から顕在化することがありますが、この発見が後手にまわると、対策が難しくなります。XDDP の変更三点セットを作成しながら“時間を置く”ことで、仕様変更についての理解が深くなり、初めは見逃していた他の部分との関係に気づきやすくなります。さらに実際のソースコードを変更していないので、先に変更した誰かがミスをしたことにならないため、メンバーを尊重しながら進むことができます。

3. レビューを意識した成果物を作成する

開発上流での品質向上にレビューが有効であることは既知の事実です。しかし、現実には設計レビューで見落とした問題が後工程に残り、開発工数全体を圧迫する原因になってしまうものです。XDDP では「レビューを意識した成果物」を作成することで、読み手側も書き手側も、問題に気づきやすくなることを目指しています。

4. 上流で問題に向き合い、無駄を排除する

「無駄を排除」する方法として、XDDP では短納期におされた過敏な対応を抑制しています。例えば、短納期の開発では変更箇所の調査中に即コードを変更する行為が見受けられます。一見これは素早い対応に思われますが、他の仕様との衝突で手戻りが発生しては無駄な工数となってしまいます。XDDP は、従来発生していたこのようなデグレードを大幅に減らすものです。XDDP を適用した上で発生してしまった問題は、製品特有の問題や組織の特徴から起こる問題と認識し、個別に対策を検討することを推奨しています。あらかじめ大幅に手戻りを減らせば、残った問題に対応する余裕も持てるものです。

XDDP はその手法が目立つため、導入の際に本質を理解せずに形だけ真似て、誤解されてしまうケースもあるようです。悩んだ時や不安になった時などは、ぜひ XDDP のポリシーを思い出していただければ、と思います。

２章：僕の XDDP 日記

<はじめに>
これは、新入社員の**山田くん**が初めての派生開発に取り組んだときのお話です。山田くんが二ヶ月の研修を終えて、ＸＸ開発部に配属されてから、しばらくのことでした。先輩の**酒井さん**から呼ばれて、電卓アプリケーションの変更を命令されました。

登場人物

山田くん
今春入社した、まだまだ素直な新入社員。今回、入社後初めてソフトウェアを開発することになった。

酒井さん
山田くんの職場の先輩。山田くんの教育担当なんだけど、自分の仕事が忙しいので、山田くんをあまりかまってあげられない。 でも本当はやさしい先輩？

清水課長
山田くんと酒井さんの上司。時に厳しく時にやさしく指導する頼りになる課長。

６月１８日（月）

今日、先輩の酒井さんに言われて、電卓アプリの変更をやることになった。

やってほしいのは、計算結果の保存と、１０進数と１６進数の変換、マイナスの数値への対応だということだった。

二週間もあればできるでしょう、って言われてソースファイルを渡されて、大学の時にやっていたように、読みながら修正をかけようとしていたら、清水課長がそれを見ていて、そんなやり方じゃあダメだよ、と言われてしまった。(>_<)

まずは、何を変更したいのか、そのために何をするのか、をわかるようにしておきなさい、と言われて、変更要求仕様書というものの書き方を教えてもらった。この書き方は USDM というらしいけど、やりたいことと、やることを、こんな風に関連づける書き方があるのは知らなかった。

確かにソースファイルを直しているうちに、何をやりたいのか、わからなくなってしまうことがよくあったので、こうやってわかるようにしておくのはいいことだな。(^^)

6月19日（火）

今日は、もらったソースファイルを読んでいた。

ソースコードの中で何をやっているのかを確認するのと、どこをどのように直せばいいのかの検討をおおざっぱにつけてみた。

大体、どこらへんを直せばいいのかはわかったので、いつもならとっとソースコードを変えてしまうのだけど、ぐっと我慢して、ここでわかったことをメモしておいた。

昨日、いきなりソースコードを変えてしまうと、かえってできあがるのが遅くなると言われたので、ソースコードを直さないようにしているけど、先に進んでいないようで、なんとなく不安だ。

6月20日（水）

今日は、ソースファイルを読んでわかったことを元に、何を変更するのかということを変更要求仕様書に書いてみた。

ソースコードとして書いてしまえば簡単だけど、文章にしようとすると意外と書けないものだということがわかった。文章で書くとソースコードの何を変えなければならないのかが、深くわかってくるような気がした。

それと文章にしようと考えていたら、もっといいやり方に気づいた。これはソースコードをいきなり書いていたら気がつかなかっただろう。いきなりソースコードを書かないというのは、こういうこともあるから言われたのかな？

でも、こうやって書いたものとソースファイルの修正が最終的にどうつながるか、よくわかっていない気がするので、そのあたりを清水課長に聞いてみよう。

6月21日（木）

今日は、ソースファイルの修正と仕様の変更のつなげ方がよくわからなかったので、そこを聞きにいって、TM を教えてもらった。

TM というのは、トレーサビリティ・マトリクスといって、仕様を行方向に書き、ソースファイルなどを列方向に書いて表形式にする書き方だ。

この書き方を教わってやってみたら、仕様がソースファイルのどのあたりと対応しているのかを一目で見ることができて、仕様とソースファイルの関係の見通しがよくなった。

一か所のソースコードの修正ではすまなくて、いくつもソースコードの修正をかけなければならないときにこうやって書くと、どこを修正しなければならないのかを表現しやすい。仕様とソースファイルの対応がどうなっているのか、確かにイメージしやすくて便利だ。いい方法を教わった。(^^)

TMでは、変更方法を書かないで、丸をつけるか、せいぜい関数名を書くくらいの抽象度におさえておかなければならないらしい。変更方法は別のドキュメントで書くのだそうだ。あの小さな四角の中に変更方法を書くのは難しいからかなぁ。

6月22日（金）

TM に対応する箇所それぞれに、どのように変更をかけるかのやり方を書いた。こういうドキュメントは変更設計書というらしい。

変更設計書にソースコードをそのまま書いたらいけないそうなので、文章で書くようにした。これは変更要求仕様書のときと同じ注意だ。

やり方を文章で書いていたら、変更方法でまずいところに気がついた。これは最初のときには気がつかなかったので、ソースコードをいきなり修正していたらバグになっていただろう。

こうやって書いていくのは、ソースコードに何度も見直しをかけるのと同じようなことになるのかも知れない。やったことはないけど、机上デバッグというのがこんなものかも。

こんなにずっとソースコードを書かないでプログラムを作ったことはなかったけど、これは最後の最後まで見直しができるように、ということを意識させているのかな？
ソースコードの修正をかけるときにどんな感じがするのか、ちょっと楽しみだ。(^^)

6月25日（月）

今日は、変更部分に関するレビューをやった。

酒井さんが「山田くん、どこまでできた？」と聞いてきたので、今こんな状況です、と話したら、それじゃあレビューをして確認しようという話になった。

作った変更要求仕様書と変更設計書を元に説明をしたら、TM のいくつかで、抜けのありそうな箇所を指摘された。指摘されたところは、自分では気がつかなかったので、ソースコードを書く前に指摘されてよかったと思った。

自分では完璧だと思ったけど、こうやって見てもらうというのは大事だ。あんまり面倒をみてくれなかったので、オン・ザ・ジョブ・トレーニングじゃなくて、お任せ・ジョブ・トレーニングじゃないかって、ちょっと不満に思っていたけど、やっぱり流石だな。

それとレビューをやって思ったけど、ソースファイルのレビューをやるより、仕様書や設計書といったドキュメントのレビューをやる方が、効率的な気がする。

文章だからわかりやすいというのもあるのだろうけど、全体を見通せるような文書があるというのが効いているのかもしれないな。

6月26日（火）

今日は、ソースコードの変更を行った。

レビューで指摘を受けた部分を修正して、酒井さんに確認したら、これでいいわよと言われて、ソースコードを変更することになった。

変更要求仕様書も変更設計書も書いて、レビューもやって、という状態で書くと、こんなにサクサク書くことができるものだとは思わなかった。

変更設計書に詳細な変更方法を書いているので、いつもと違ってソースコードの変更で悩むところがなかった。

ソースコードを書くときに、迷わないで書いていけるというのは気分がいいな。(^^)

それにしても、一日でソースコードの修正が済んでしまうとは思わなかった。なんていうか、先にソースコードの修正をかけるよりも、こういうやり方のほうが早いといわれたことの意味がわかったような気がする。

6月27日（水）

今日は、ユニットテストをやって関数の確認をしてから、変更部分を含めた全体の動作確認をした。

修正なしで、一発で動いた。これは自分でもすごいと思う。(^^)

今までは、ソースコードを書いた後に、動かしては直しての繰り返しをやるのが当たり前だと思っていたけど、こんな風にきちんとやっていくと直すのがほとんどいらなくなるんだ。

変更内容がほぼ頭に入っているので、修正も簡単に済んだ。ユニットテストを書いている最中に単純ミスに自分で気づいて直せてしまうところもあって、何度も見直しをかけることの効果を実感した。

電卓アプリの設計書も書いておいてくれと言われたので、後は今回ソースファイルを調べたことや変更要求仕様書、変更設計書とかを元に設計書を作れば終わりだな。酒井さんには二週間と言われていたけど、それより早く終わりそうだ。

なんか、レベルが上がったような気がして満足感がある。今回やった電卓アプリの修正には「達成感」がすごくあったような気がするな。(^^)

3章：How to XDDP（概要）

※ 文中の「テキスト」は『「派生開発」を成功させるプロセス改善の技術と極意』(清水吉男　著)のことです。

（1）　二つのプロセス

派生開発には、一般に「機能変更要求」と「機能追加要求」の２種類の要求があります。

「機能変更」と「機能追加」とでは要求の性質が大きく異なるため、要求仕様書の書き方を変えます。また、プロセスも『**変更用プロセス**』と『**追加用プロセス**』に分けます。それぞれの使い分けを整理したのが次の表です。

<table>
<tr><th colspan="2"></th><th>対応</th><th>対応する要求仕様書</th><th>対応プロセス</th></tr>
<tr><td rowspan="4">機能レベル</td><td rowspan="2">追加</td><td rowspan="2">機能追加
として扱う</td><td>追加分：
追加機能要求仕様書</td><td>追加用プロセス</td></tr>
<tr><td>変更分：
変更要求仕様書</td><td rowspan="5">変更用プロセス</td></tr>
<tr><td>移植</td><td>通常は
機能変更
として扱う</td><td rowspan="4">変更要求仕様書</td></tr>
<tr><td>削除</td><td rowspan="4">機能変更
として扱う</td></tr>
<tr><td rowspan="3">仕様レベル</td><td>追加</td></tr>
<tr><td>変更</td></tr>
<tr><td>削除</td></tr>
</table>

※ プロセスの使い分けについての説明は、次頁に記載します。

機能変更の場合、『**変更用プロセス**』を用いて、変更分の要求仕様書（『**変更要求仕様書**』）を作成します。

機能追加の場合、『**追加用プロセス**』を用いて、追加分の要求仕様書（『**追加機能要求仕様書**』）を作成するだけではなく、『**変更用プロセス**』を用いて、変更分の要求仕様書（『**変更要求仕様書**』）も作成します。

機能追加とは言え、既存システムへの追加ですから、必ず既存部への変更が発生します。

『**変更要求仕様書**』には、次の点を特に注意して記載します。

① 追加機能を埋め込む部分に対する変更（例：既存のメニュー画面に追加機能を起動するためのボタンを追加する）

② 既存機能への影響（例：リソースやデータの共有）

『**変更要求仕様書**』、『**追加機能要求仕様書**』を作成して顧客に確認し、顧客と合意することにより、「仕様の誤解」問題をクリアにします。

プロセスの使い分けについて

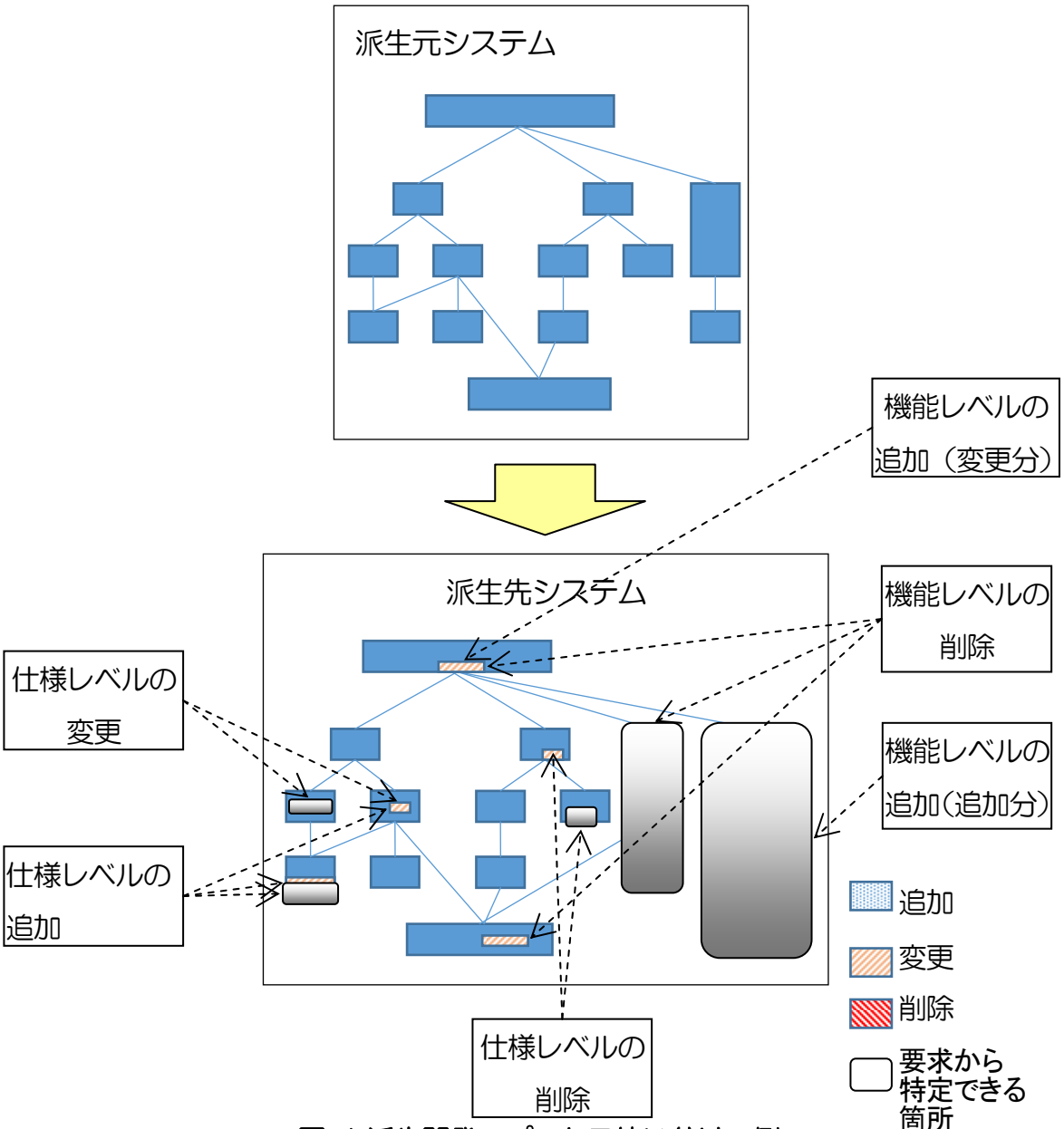


図 1 派生開発のプロセス使い分けの例

- 仕様レベルの追加・変更・削除

既存のソースコードのクラス／関数等で仕様を実現している箇所の一部に対して、処理やデータを追加・変更・削除することで「**機能変更要求**」に対応する場合が該当します。おもに既存のクラス／関数やデータの仕様について対応を行うため、『**機能変更**』として扱います。

・ 機能レベルの追加

「**追加機能要求**」を実現する新規クラス／関数を作成することで対応する場合が該当します。新規クラス／関数は、『**機能追加**』として扱います。新規関数の呼び出し元や、追加機能に関連するデータの変更が生じる箇所は、『**機能変更**』として扱います。

・ 機能レベルの削除

機能を実現しているクラス／関数等を削除することで「**機能変更要求**」に対応する場合が該当します。削除対象の関数を呼び出している箇所や、関連データの削除等、削除対象のクラス／関数以外に変更が生じるため、『**機能変更**』として扱います。

・ 移植

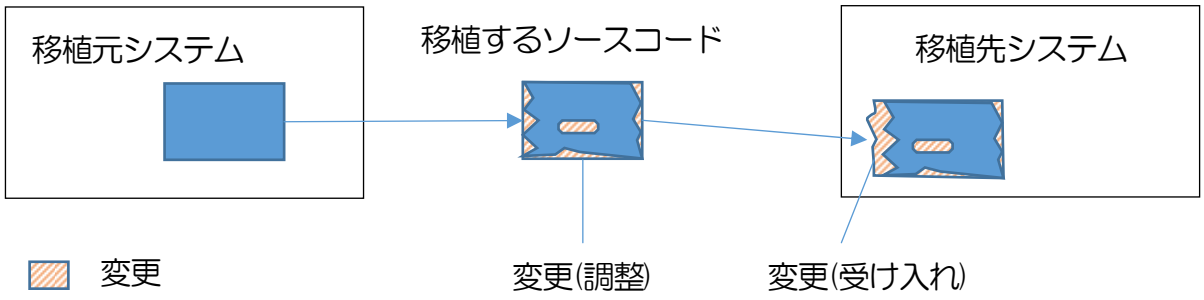
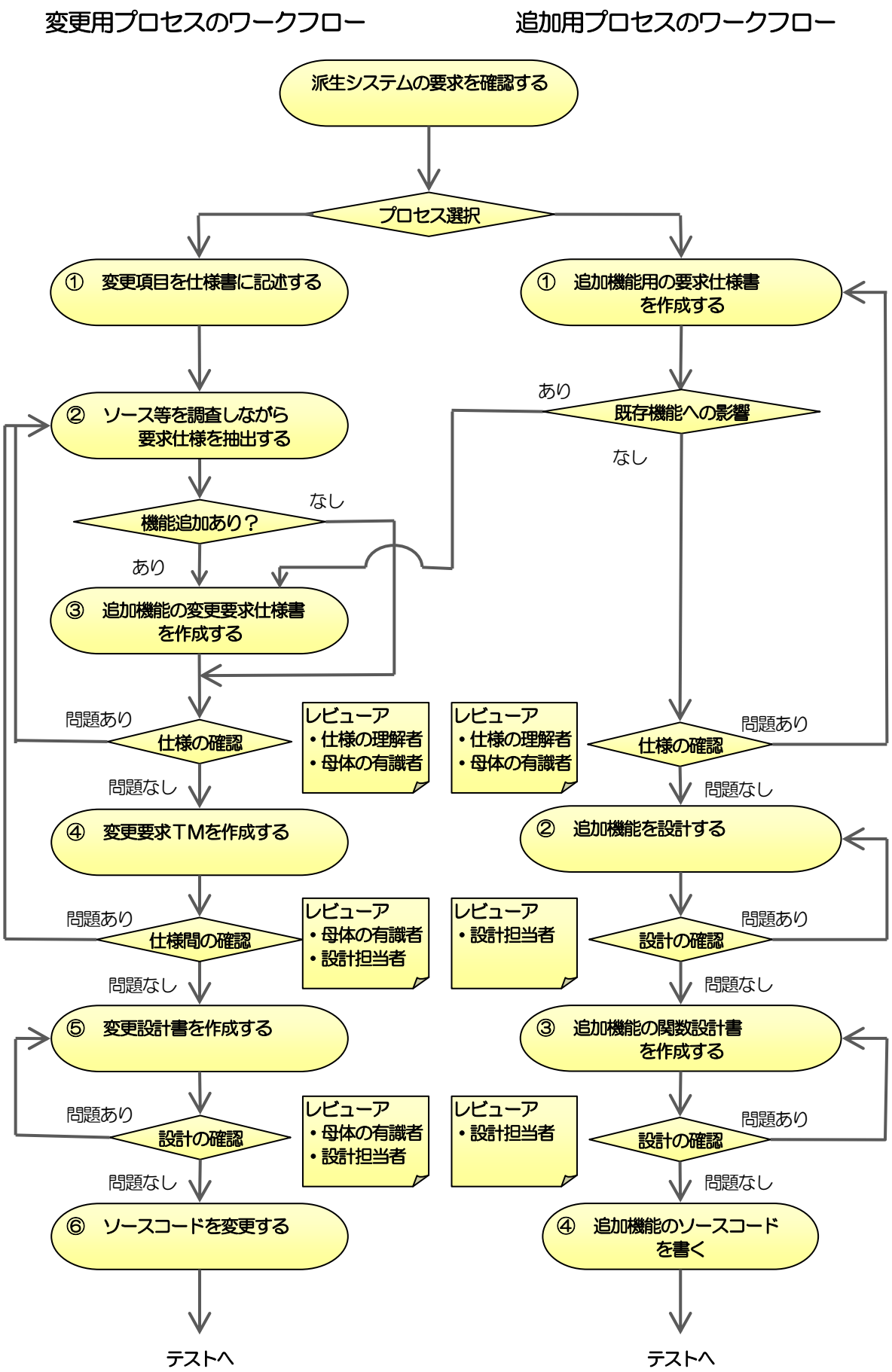


図 2 移植の例

他のシステムで動作している機能のソースコードを派生開発の対象システムで動作させることで「**機能変更要求**」に対応する場合が該当します。移植するソースコードは、移植先システムのアーキテクチャ等の影響により変更が必要な場合が多いため、『**機能変更**』として扱います。移植先システムでは、移植するソースコードの受け入れのための変更が生じるため、『**機能変更**』として扱います。

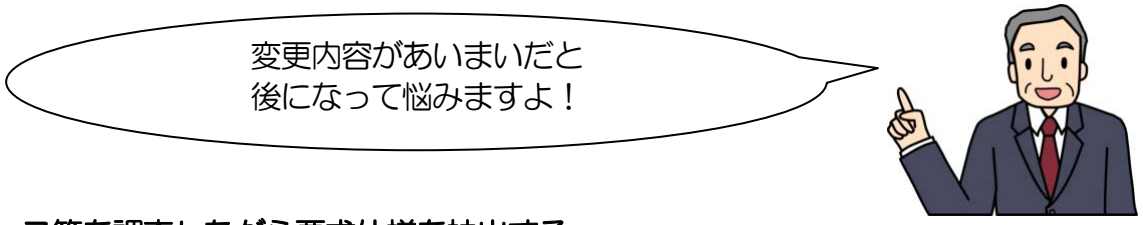


（2） 変更用プロセス

① 変更項目を仕様書に記述する

- 顧客の変更要求をもとに、要求項目毎に『USDM[1]』を用いて『**変更要求仕様書**[2]』を作成します。

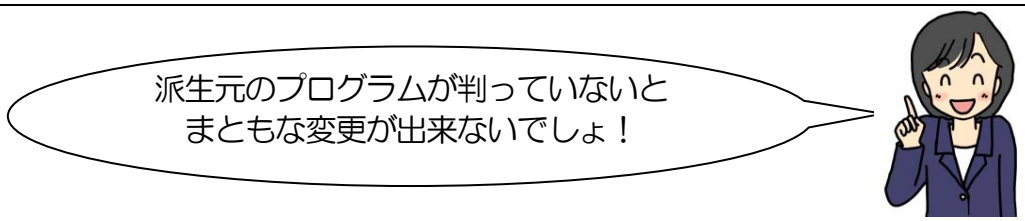
目的	変更項目を明確にして視覚化することで、変更内容の抜け・漏れを防ぎます。



② ソース等を調査しながら要求仕様を抽出する

- 既存の文書やソースコードの情報を調査し整理します。
  - 関連する機能仕様書、各種設計書、関数仕様書を集めて一覧にまとめます。
  - 気がついた内容を書き留めておきます。
- 変更仕様に対応するソースコードを探し確認します。
  - ソースコードを読んで、仕様・構造・動作を理解し、調査結果を書き出します。（スペックアウト）
  - 変更要求に対する変更箇所（変更仕様）を探す際には、「データ構造」「処理構造」「制御構造」の情報を調査します。
- システムのアーキテクチャを確認します。

目的	派生元のプログラムを理解することで、的確な変更の実現を より確実なものにします。



---

[1] Universal Specification Describing Manner. ソフトウェア開発における要求を仕様化する方法。詳細は「要求を仕様化する技術・表現する技術」(清水吉男　著)を参照。

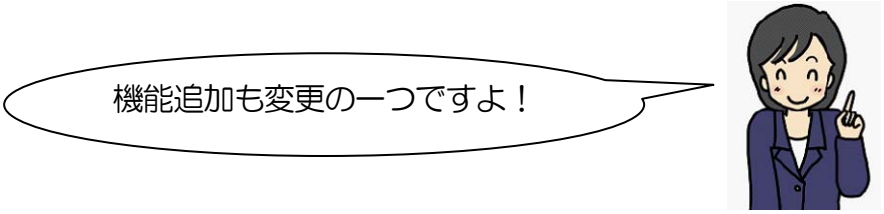
[2] “変更してほしいこと(Change Requirement)”について、関係者の間で特定(Specify)できたことがまとめられた文書。詳細はテキスト 2.3「「変更要求仕様書」の導入」を参照。

③ 追加機能の変更要求仕様書を作成する

※　機能追加がない場合は、③をスキップし、④へ進みます。

- 派生元のプログラムに機能追加を受け入れるための変更要求仕様（追加機能の接合方法）を抽出し、『USDM』を用いて『**変更要求仕様書**』を作成します。
- 追加機能について、『**追加機能要求仕様書**[3]』に記述します。
- 追加機能の受け入れ箇所を、『**変更要求仕様書**』に追記します。
- 二つの文書の連携を確認します。

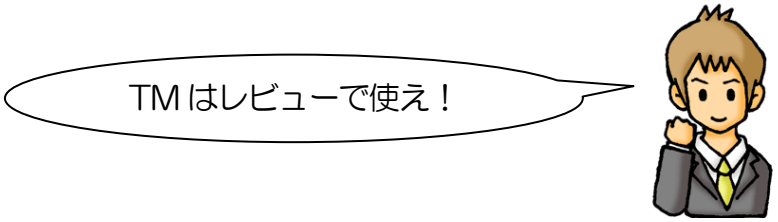
目的	追加機能との切り口を明らかにします。機能追加も変更の一種と捉え、変更の抜け・漏れを防ぎます。



④ 変更要求TM[4]を作成する

- 『行』方向に変更要求仕様もしくは変更要求仕様の項目名を、『列』方向にタスクやマクロ名を記載します。
- 変更がある場合、交点にマークを入れていきます。
  - 機能仕様書や設計書から変更すべき仕様を発見したときに、マークを入れます。
  - ソースコード上の該当箇所を発見したときに、マークを入れます。
  - レビューなどの場で指摘されたときに、マークを入れます。

目的	どこを変更するのかを明確にすることで、レビューをしやすくし、バグを無くします。また、作業の衝突箇所を明確にし、後戻り工数を無くします。



---

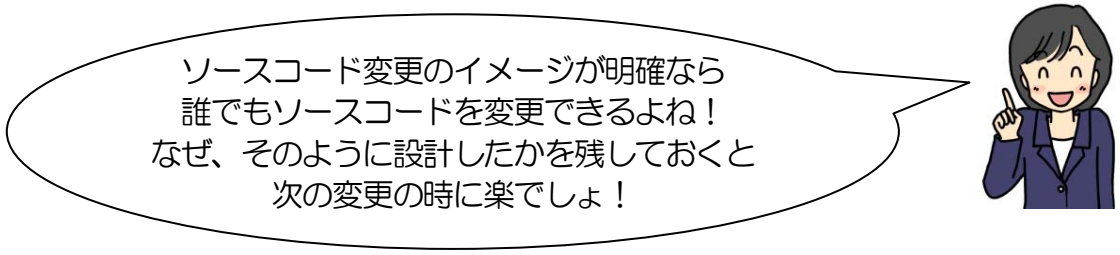
[3] 追加される機能ごとに作成される要求仕様書であり、変更要求仕様書の付属文書として位置づけられる。詳細はテキスト 2.5.2「追加機能要求仕様書」を参照。

[4] 変更要求とトレーサビリティ・マトリクスを組み合わせ、要求とソースコードの変更箇所を紐づける文書。詳細はテキスト 2.5.3「トレーサビリティ・マトリクス」を参照。

⑤ 変更設計書を作成する

- TM の交点毎に『**変更設計書**[5]』を作成します。
- 変更設計書の構成：
  - 変更仕様などの位置情報（ヘッダー情報）
    変更要求仕様、変更仕様番号、モジュール名など、TM 上の「位置」を示す情報。
  - 構造の変更
    データ構造や処理構造が変化するときに、図で変更の様子を示す。
  - 定義の変更
    関数の外の定義文の変更を記述する。
  - 関数内の変更
    クラス：関数の中の変更を記述する。
  - 変更後の確認項目
    変更したときに確認する項目（ホワイトボックステストに相当）を記述する。

目的	変更設計書を作成することで、ソースコード変更のイメージを明確にします。 また、設計の意図を残します。

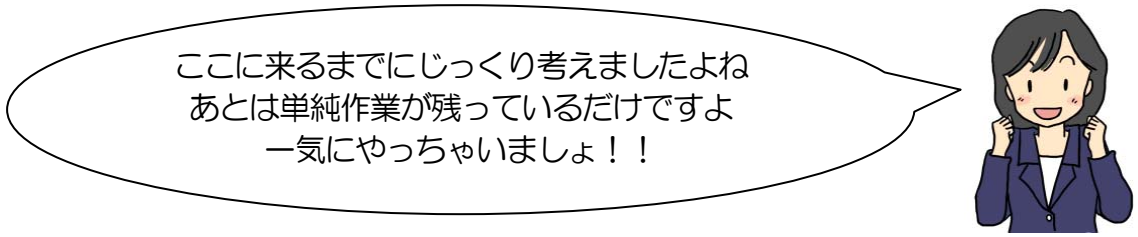


[5] 変更箇所や変更の意味を文書にしたもの。詳細はテキスト 2.5.5「変更設計書」を参照。

⑥ ソースコードを変更する

- 変更設計書をソースファイル単位に集めます。
- 変更設計書に従って、**一斉に**ソースコードを変更します。
- もし⑤が完了するまでに仕様変更が生じても、⑥開始までは変更対象のソースコードに手を付けていません。従って仕様変更に起因するソースコードの再変更工数は発生しません。

目的	ソースコードの変更作業を、変更設計書からの単純な変換プロセスとしてしまうことで一定の生産性の確保が可能になります。 単純な変換プロセスとしたことで、要員の追加投入も可能になります。



（3）　追加用プロセス

追加プロセスは、基本的には新規開発のときのプロセスと同じです。

① 追加機能用の要求仕様書を作成する

- 追加機能の要求と仕様を『**USDM**』を用いて『**追加機能要求仕様書**』を作成します。
- 既存機能の仕様に変更が必要な場合は、『**変更要求仕様書**』に記載します。
  （変更用プロセス③）

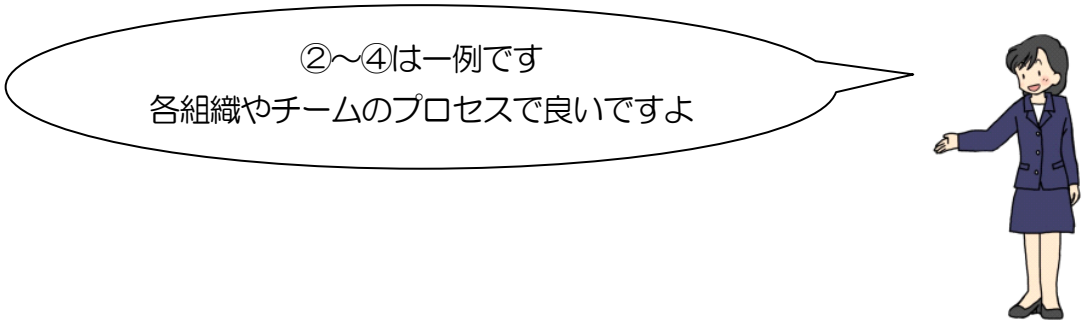
目的	この段階で既存機能との仕様調整をすることで、追加機能の設計を変更と切り離して進めることが可能になります。 設計以降の段階からの手戻りも減らすことが可能になります。

②～④は、各組織やチームの開発方法に合わせて実行します。

② 追加機能を設計する

③ 追加機能の関数設計書を作成する

④ 追加機能のソースコードを書く



Appendix A:用語辞書

本文書における、用語の定義を以下にまとめます。

用語	意味
【アルファベット】	
XDDP	［eXtreme Derivative Development Process］ 派生開発に特化した開発方法論。「短納期」や「部分理解」といった派生開発特有の問題に合理的に対応する方策として、XDDP では ・ 変更要求仕様書 ・ TM（Traceability Matrix） ・ 変更設計書 の成果物３点セットを作成することで、効果的なレビューの機会を確保し、開発プロセスを進める方法を提唱する。
PFD ※ サンプル添付	［Process Flow Diagram］ プロセスと成果物の連鎖を記述する表記法。構造化分析手法で用いる DFD（Data Flow Diagram）を元とする。開発プロセスを設計するために使用する。 PFD では「成果物」と「プロセス」の〝関係〟を表すだけで、〝順序〟については表記しない。そのプロセス遂行に必要な入力は何か、そのプロセスでは何を出力するか、のみを示す。PFD で開発プロセスを可視化することで、その成果物が、どのプロセスで作られ、どのプロセスで使われるかが明確となり、最適なプロジェクトの計画を設計する。
USDM ※ サンプル添付	［Universal Specification Describing Manner］ 要求仕様の表記法。要求と仕様を階層的に表現する。 USDM では、機能に関する「要求」は“振る舞い”を表現し、「仕様」を引き出す役割を担うという考えから、「要求」と「仕様」を表現上使い分ける。また、仕様の抜けや漏れを防ぐことを目的として、要求の背景としての“理由”を記述する。
TM ※ サンプル添付	［Traceability Matrix］ 「変更要求仕様書」と「変更設計書」とのトレーサビリティを確保するためのマトリクス形式文書。 TM では、“変更要求”に対する影響範囲を可視化するため、以下のような特徴で記述する。 ・ マトリクスの行には、“変更要求仕様”を記述する。 ・ マトリクスの列には、列にはソースファイルやタスクなどの機能をプログラムコードに実装したまとまりの単位で記述する。 ・ 行（仕様）と、列（実装単位）とが紐づいていたら、マトリクスの交点にマークをつける。

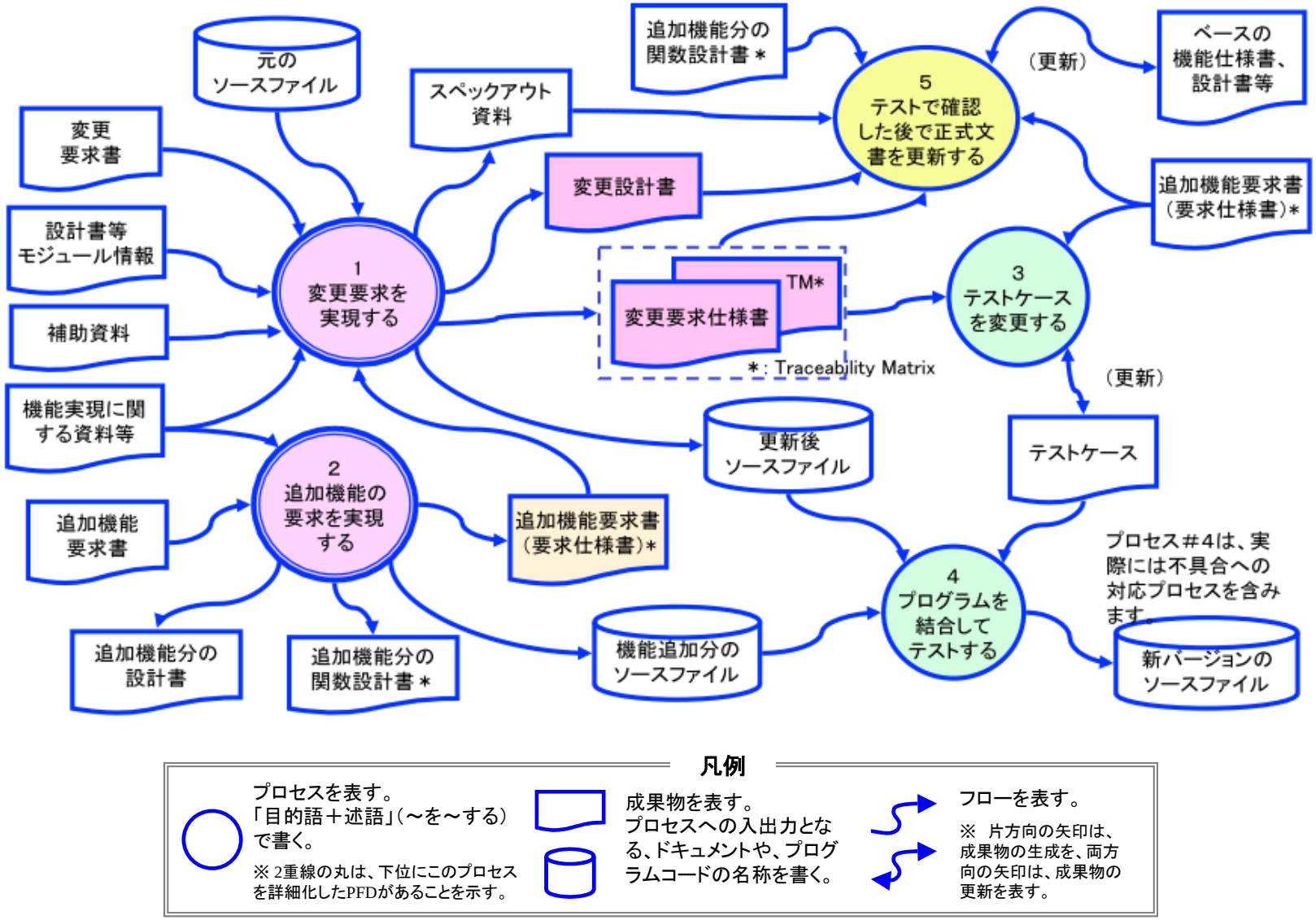
用語	意味
【か行】	
機能仕様書	開発対象システムについて、どのような機能を提供し、どのように振舞うかといった、外部視点で機能を説明した文書。 機能だけではなく、品質や性能などを織り交ぜて記述されていることもある。
【さ行】	
仕様	要求を満たすための具体的な振る舞いの記述。 仕様であるための条件として、 ・ 関係者によって合意（Specify）できている ・ 具体的な振る舞いが明確である（コード化可能） ・ 検証可能である の３点を満たしていることが重要である。
スペックアウト	派生元となるシステムのプログラムコードを解析し、必要な設計情報を生成する行為。派生元に対する理解が十分でない場合、派生開発プロセスの前準備として実施される。 スペックアウトの目的としては、変更対象箇所の特定や、リファクタリングのため、アーキテクチャ改修のためなどがある。それらの目的に合った範囲を調査・理解し、プログラムの構造や設計意図などをスペックアウト資料として作成する。
成果物	開発プロセスにおいて作成された文書、もしくはプログラムコード。
【た行】	
追加機能要求仕様書	派生開発における、機能追加に関する要求仕様書。 XDDP では、派生開発の要求仕様書は「変更要求仕様書」に記載し、追加機能要求仕様書は、その下位文書にあたる。 機能追加が必要な開発では、 ・ 既存システムのどこに機能追加するか（機能追加を受け入れるための変更）を「変更要求仕様書」に記述 ・ 追加する機能に関して、「追加機能要求仕様書」に記述 の２種の仕様書を併用して定義する。
【は行】	
プロセス	作業工程、および工程内で行われる作業項目。
部分理解	派生開発において、派生元となるシステム全体を理解するのではなく、期間やコストなどの理由により、今回の変更にかかわる部分に限定した理解により、開発を進めること。

用語	意味
変更設計書	派生開発において、具体的な変更箇所を記述した文書。原則として、TM のマトリクス交点にマークした箇所に対応して作成する。 変更設計書は、以下のような特徴で記述する。
※ サンプル添付	・ 具体的な変更方法をできるだけ文章で記述する。 ・ 変更する“差分”の記述に徹する。 ・ How の視点で書く。
変更要求	要求（Requirement）の一種。ベースとなるソフトウェアやシステムがあり、その一部もしくは全体を変える（部分的な機能削除や、機能追加を含む）ことで実現したいことを表す。
変更要求仕様書	派生開発において、今回の開発で変更したいこと（Change Requirement）について、関係者が変更内容まで含め、その内容について特定 （Specify）したことをまとめた文書。 変更要求仕様書は、以下のような特徴で記述する。 ・ 変更を“変更要求”と“変更仕様”の階層体系でとらえる。 ・ 変更要求も変更仕様も必ず“before”、“after”で表現する。 ・ What（および Why）の視点で書く。

【や行】

要求	ソフトウェアやシステムを利用することで実現したいこと（Requirement）。
要求仕様書	システムを開発する際に、そのシステムで実現したい事柄を記述した文書（機能仕様書と同義となっていることもある）。

PFD 表記例　：　派生開発の変更と追加を統合した全体の PFD



出典：AFFORDD勉強会資料　派生開発アプローチ：XDDPの詳細

USDM 記述例　：　「荷物配送システム」 要求仕様書

			要求／要求仕様		
	ユースケース①		集荷依頼を配送担当者に送る		
集荷依頼	要求	Cont10	荷物の集配依頼を受けて、担当地域の配達員の端末に住所や氏名等の情報を送る		
		理由	配達中の配達員に情報を送ることで、配達ルートの中で荷物を集めたい		
		説明	依頼者からの電話を受けて、集荷先の氏名や住所、メールアドレスなどを聞き出して入力したデータが画面上に表示している		
		要求	Cont10.1	集荷先情報入力画面切り替えて不足する情報を入力する	
			理由	電話を受けながら住所等を入力しているので、それを転用したい	
			説明		
			＜集荷情報入力画面の表示＞		
		□□□	Cont10.1.1	「集荷指示ボタン」の押下で「集荷情報入力画面」に切り替える	
		□□□	Cont10.1.2	顧客の電話から入手した以下の情報を、「集荷情報入力画面」の対応する欄に移動させる ・集荷指定日時 ＜集荷先情報＞ ・氏名 ・住所 ・郵便番号 ・電話番号 ・集荷時特記事項	
		□□□	Cont10.1.3	「指示番号」は、システムが自動で設定しているデータを入手して表示する	
			＜不足情報の入力＞		
		□□□	Cont10.1.6	集荷指定日時が「空欄」の場合は、社内規定に従って電話を受けた時間から割り出した時間を入力する	
		要求	Cont10.2	依頼者の住所を担当する配達員を探して、集荷指示データを配達員の端末に送信する	
			理由	地域によって配達員が決まっていて、配達中のことが多い	
			説明		
			＜担当配達員の検索＞		
		□□□	Cont10.2.1	住所を入力して「担当配達員を探す」ボタンを押す	
		□□□	Cont10.2.2	見つかった配達員名を表示する	
		□□□	Cont10.2.3	担当配達員が複数表示された時は、そこから1名を選ぶ	
			＜配達員情報の表示＞		
		□□□	Cont10.2.6	配達員データの「受け持ち地域番号」から「地域名」を入手する 【説明】受け持ち地域番号から、受け持ち地域の番地まで割り出す処理は別に用意されている	

出典：AFFORDD勉強会資料　USDM入門

TM 記述例　：　変更要求仕様書に関する変更点一覧

			XDDP事例(変更要求仕様) 変更要求仕様の表現や機能追加要求仕様書との連携の様子などを参考にしてください		TM 機能分類	起動部	StartUp.src	StartDvcChk.src	メイン操作画面	MainDispOpe.src	画面操作設	MainSetOpe.src	GrafSet.src	MeasPntSet.src	印刷装置設定画面	計測画面	MainMeasOpe.src	CalGrafTyp1.src	Grafdsn.src
カテゴリ名 (記号)	要求	CAL05 理由 説明	受信したデータをリアルタイムに表示しているところで画面がいっぱいになる毎に表示内容を印刷する機能を追加する 紙に印刷することで、過去のグラフと見比べたい 表示機能は既に実現している状況で、印刷機能だけを追加する																
		要求	CAL05.1	計測情報の設定画面で、印刷装置等の設定を行うための操作画面を表示するボタンを追加する 計測情報の設定操作の一環として操作させたい	追加[PRT01.1]に対応														
			＜選択ボタンの追加＞																
		□□□	CAL05.1.1	計測情報の設定画面の「計測ポイント」の下に「印刷情報の設定」ボタンを追加する					○		Fa()							
			＜印刷情報設定画面の表示＞																
		□□□	CAL05.1.5	「印刷情報の設定」ボタンが押されたときは、「PRT01.1」で設定される処理の呼び出しを追加する					○			Fb()						
		要求	(要求番号)	計測モードに入ったところでバッファーを確保する処理を追加する	追加[PRT01.2]に対応														
			＜バッファ獲得のタイミング＞																
		□□□		メイン操作画面から「計測モード」ボタンを押下を検出後、計測モードの画面を表示する前にバッファ獲得処理を追加する 【説明】実際の獲得処理は追加機能要求仕様で実施											○	Fc()		
		要求	CAL05.2 理由 説明	グラフ表示領域がいっぱいになった時点で印刷する処理を追加する 印刷を開始するのはこのタイミングしかない 追加機能要求仕様番号[PRT01.2]に対応	追加[PRT01.3]に対応														
			＜通常時の印刷処理の追加＞																
		□□□	CAL05.2.1	表示用の領域がいっぱいになった時点で「PRT01.2」で設定される印刷処理の呼び出しを追加する											○			Fd()
			＜計測終了時の印刷処理の追加＞																

出典：AFFORDD勉強会資料　XDDP入門

変更設計書記述例　：

変更設計書の構成（ヘッダ情報）

変更設計書

プロジェクト名	XDDP プロジェクト	作成日	2007 年　7 月　10 日		
ソース名／ タスク名	RectCalPart1.c	作成者			
		修正者			□修正
変更要求仕様	計算の基準が設定値だけの状態から、一時的に基準値を変更できるようにする	確認者			□確認
#CAL_01.2		見積り行数	35	見積り時間	30
		変更行数		作業時間	

■修正方針

特になし

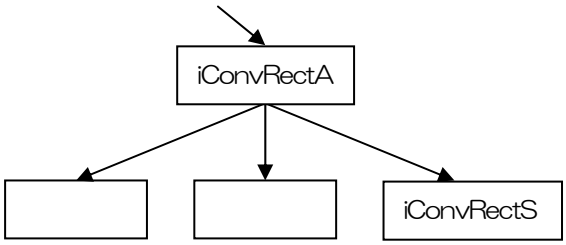
■データ構造の変更

なし

変更方法:
プログラム構造・データ構造・関数など、変更方法の詳細を記述

■関数呼び出し構造の変更

iConvRectAll　のモジュールの凝集度が悪化するのを回避するために、この中の処理を一部分離し、新しく（iConvRectSub）の関数を設けて、そこに関連する処理を移して、基準値の変更処理をそこで実現する。その他の下位モジュールには変更は無い。



■関数外の変更

項目#	変　更　内　容	予想行数	
1	関数名変更（iConvRect　→　iConvRectAll）にともなって定義を変更する	1	□
2	分離独立する関数（iConvRectSub）の定義を追加する	1	□

■関数の変更

関数名	iConvRectAll (Vert, Rect, Base)		■変更、□追加、□削除

変更内容：

項目#	変　更　内　容	予想行数	
1	関数のパラメータに、前回の基準値を表す情報（Base:int）を追加する。	1	□
2	計算前に、測定関係設定情報（TblSetInf）の「Kijun」から基準値を取り出している処理を切り出して、新しい関数（iConvRectSub）に分離し、Base の値をその関数に引き継ぐ。関数からの戻り値を新しい「基準値」として計算する。その後の計算式に変更はない。	7	□

確認項目：

項目#	確　認　内　容	チェック
	iConvRectSub の関数を繋いだ状態でテストを行う	
1	設定情報の Kijun=35, Vert=120, Rect=25, Base=0 としたとき、基準値として Kijun の値が採用されて関数の戻り値が「1230」となる	
2	設定情報の Kijun=35, Vert=120, Rect=25, Base=45 としたとき、基準値として Base の値が採用されて関数の戻り値が「1375」となる	

変更後の確認項目

出典：AFFORDD勉強会資料　XDDP入門

変更履歴

第一版	２０１２年　５月２５日	初版
第二版	２０１３年　５月２４日	全体的に変更

参考文献

「派生開発」を成功させるプロセス改善の技術　　清水吉男　著　　技術評論社
ISBN978-4-7741-3249-5　C3055

[改訂第２版] 要求を仕様化する技術・表現する技術　　清水吉男　著　　技術評論社
ISBN978-4-7741-4257-9　C3055

アンケートご協力のお願い

T3 研究会では、本冊子の継続的改善のため、皆様からのコメントをお待ちしております。
アンケートにご協力いただける場合は、以下 URL よりご意見、ご感想をお寄せください。

http://enq-maker.com/12UVmB8