XDDP Patterns—A Pattern Language for eXtreme Derivative Development Process—

NORIKO KAWAGUCHI, KOBELCO SYSTEMS CORPORATION

The development method for adding new functions into or modifying existing software systems, known as “Derivative Development”, is quite different from that for the development of new software systems. A development process dedicated to the Derivative Development, named as “eXtreme Derivative Development Process (XDDP)”, has been proposed and widely applied to embedded system development in Japan. In this paper, “XDDP Patterns” is introduced based on the XDDP practices from the viewpoint of Pattern Language. The pattern map of XDDP Patterns is presented as “XDDP Route Map”, inspired by the railway maps of Osaka and Kobe area in Japan. This pattern map would be used as a guide map to make XDDP practices easier. The work for this paper consists of my previous paper “Proposal of XDDP Route Map for Disseminating XDDP” (Kawaguchi 2017) and 3 new patterns of USDM, Reverse Engineering, Code Change.

Categories and Subject Descriptors: **・Software and its engineering~Software creation and management~Software post-development issues~Software evolution** ・Software and its engineering~Software creation and management~Software post-development issues~Maintaining software ・Software and its engineering~Software creation and management~Software development process management~Software development methods

General Terms: Human Factors

Additional Key Words and Phrases: Patterns, Pattern Language, Software Development Process, Derivative Development, XDDP, USDM

ACM Reference Format:

Kawaguchi, N. 2020. XDDP Patterns—A Pattern Language for eXtreme Derivative Development Process—. HILLSIDE Proc. of Asian Conf. on Pattern Lang. of Prog. 9 (September 2020), 16 pages.

1. Introduction

筆者は、「eXtreme Derivative Development Process（以下、XDDP と記す）」という派生開発に特化した開発方法論を用いて、所属企業の開発部門に対して派生開発現場のプロセス改善支援を行っている。派生開発とは、新規開発に対峙させた概念で、既存のベース・システムに対して、製品価値を高めるために機能追加や変更を行うことを指す。

筆者は、2008 年当時、派生開発における不具合や手戻りの多発に苦労していた。そして、その時期に XDDP に出会い感銘を受け、この開発方法論が自身の現場にも有用であると感じた。そこで、担当案件に XDDP を導入し、不具合や手戻りを低減させることができた。その後、2016 年からは、自身の経験を活かして、XDDP を用いた派生開発プロセス改善支援を行っている。

しかし、XDDP は、その有用性にも関わらず、現状では広く普及するまでには至っていない。筆者は、これまでの XDDP 導入経験から、XDDP を普及させるには、以下の課題に取り組む必要があると考える。

① 多様なステークホルダーが理解可能なガイドマップ

XDDP 導入にあたっては、開発者やプロジェクト・マネージャー、ビジネス・アナリストといった多様なステークホルダーから理解とコミットメントを得る必要がある。それゆえ、XDDP 普及には、これらのステークホルダーが XDDP を容易に理解できるようにするためのツールが必要である。具体的には、特定の知識がなくても XDDP プラクティスの構造とつながりが直感的に理解できるようなガイドマップの提示が必要であると考える。

また、一般に、開発現場は XDDP という未知の手法に対して抵抗感を抱きがちである。しかし、プロセス改善は、何か一つでも良いものを取り入れようという能動的な姿勢がなければ進まない。ゆえに、ガイドマップを見るステークホルダーに「肯定眼」を常に意識してもらう工夫も必要である。

② 選択的・段階的導入の手順を表したロードマップ

派生開発では、新規開発の標準プロセスを使っている現場も多く、一定レベルのプロセスは確立していると思われる。しかし、問題が起きている現場では、新規開発の標準プロセスに対して、派生開発特有の制約を考慮したテーラリングを行わずに、そのまま無理に使っていることが多い。そのような現場に対しては、XDDP プラクティスから現場の問題解決に効果があると見込まれるものを選択し、標準プロセスに組み込むことが現実的な施策となる。一方、すべての XDDP プラクティスを一気に導入するのは、現場の負担が大きく、現実的な施策ではない。

以上より、XDDP の普及には、必要最小限の XDDP プラクティスを見極めて、導入できるようにすることが必要である。具体的には、XDDP プラクティスの選択的・段階的導入の手順を表したロードマップの提示が必要であると考える。

そこで、本稿では、XDDP 普及のために、XDDP にパターン・ランゲージの考え方を応用した XDDP Patterns を提案する。

2. What is XDDP?

2.1 XDDP: A Derivative Development Process

XDDP は、清水吉男（以下、清水氏と記す）によって提唱された開発方法論で、清水氏の組み込みシステム開発やプロセス改善コンサルティングにおける実践的ノウハウの集大成である（清水 2007a）。

清水氏は、派生開発には次のような特有の制約があるとしている。

- 短納期

  派生開発の開発期間は、短く設定されやすい。なぜなら、開発期間は、開発の規模感から決められることが多いからである。派生開発は、新規開発と違って、ベースのソースコードの規模に比例して影響調査やテストの工数は大きくなるが、開発規模は比例しないことが多い。

- 部分理解

  派生開発の担当者は、ベースのソースコードをすべて理解していない状況に置かれる。なぜなら、新規開発の終了時、通常、開発要員の大多数は離任するからである。また、新規開発時に作成された設計書が、ソースコードと一致していないことが多いことも理由としてある。

派生開発の担当者は、「短納期」と「部分理解」の制約のもとで、コード変更を強いられる。そのため、担当者の思い込みや勘違いによる不具合の混入が避けられない。XDDP は、こうした不具合を低減させる方法として、合理的な開発プロセスと、そのプロセスを効果的に行うためのプラクティス（実践）を提供している。更に、XDDP 実践者が持つべきフィロソフィー（哲学）も示している。

2.2 Process

XDDP は、結合テスト開始前まで、変更プロセスと機能追加プロセスを異なる手順で行う（図 1）。

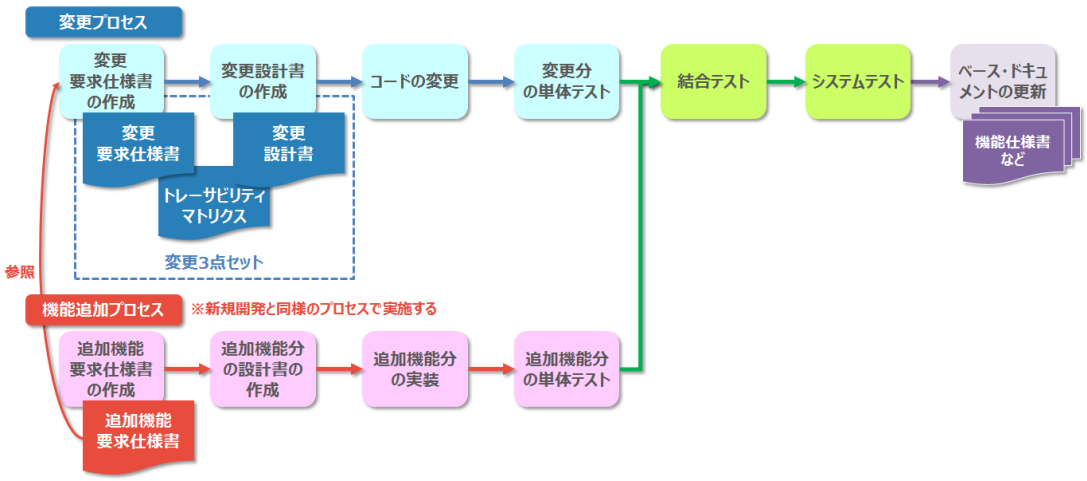

図 1.XDDP の開発プロセス

異なる手順で行うのは、既存システムに対する変更と機能追加とでは、開発者に求められることが異なるからである。変更では、既存のソースコードを抜け漏れ・誤りなく変更することが求められる。一方、機能追加では、既存のアーキテクチャを踏まえた新しい機能の要求仕様書や設計書の作成が求められる。

変更プロセスでは、システムの変更内容を異なる視点で表現した「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」（以下、「変更 3 点セット」と記す）を、段階的に作成・レビューする（表 1）。段階的な作成・レビューにより、コード変更に着手する前に、担当者の思い込みや勘違いによる不具合を取り除く。なお、新規開発時に作成したベース・ドキュメント（機能仕様書など）は、システムテストが完了し、品質が安定した後に更新する。

表 1 変更 3 点セット

作成順	ドキュメント名	視点	記述内容
1	変更要求仕様書	What ：何を変更するか？ Why ：なぜ変更するか？	変更要求・理由（要求の背景）と変更仕様
2	トレーサビリティ・マトリクス	Where：どこを変更するか？	変更仕様とコード変更箇所の関係
3	変更設計書	How：どのように変更するか？	コードの変更方法

一方、機能追加プロセスは、新規開発と同様のプロセスで行う。ただし、追加機能を既存システムに組み入れるために行う変更は、変更プロセスで扱う。そのため、変更要求仕様書を作成する時は、追加機能要求仕様書を参照する。なお、機能追加が無い場合は、このプロセスは実施されない。

2.3 Practices

XDDP では、2.2 節の開発プロセスを支える多様なプラクティスが提供されている（章末の表 2）。当節では、「変更 3 点セット」作成において前提となる「USDM」、「スペックアウト」について述べる。

① USDM

USDM とは、「Universal Specification Describing Manner」の略称で、清水氏によって提唱された要求の仕様化技術、及び、要求仕様の記述方法である（清水 2010）。変更要求仕様書と追加機能要求仕様書は、USDM を用いて作成する。

USDM では、要求仕様を「要求」と「仕様」とに分け、階層構造で表現する。USDM の「要求」は、依頼者がシステムで実現したい“こと”を表す抽象的な概念で、仕様化の範囲を示す。一方、USDM の「仕様」は、ソースコードに変換可能な“もの”を表す具体的な概念で、要求を満たすシステムの

振る舞いや処理などを示す。また、「要求」には「理由」を付ける。「理由」は、「要求」の導出や洗練、妥当性確認の拠り所となる。

② スペックアウト

スペックアウトとは、「ソースコードを解析して設計資料を作り出す行為」（清水 2010）を指す。派生開発では、既存システムの仕様を理解するためにソースコードを読む必要がある。しかし、一般に、他人の書いたソースコードから仕様を読み取ることは難しいため、派生開発の担当者はソースコードの理解を誤ることがある。この問題を解消するには、担当者の理解状況を可視化して、第三者が理解状況を確認できるようにする必要がある。スペックアウトは、この可視化のために行う。

「USDM」、「スペックアウト」と前節で述べた「変更 3 点セット」は XDDP の主要プラクティスである。これらのプラクティスは、派生開発で発生しやすい不具合（問題）を解消する働きを持つ（図 2）。

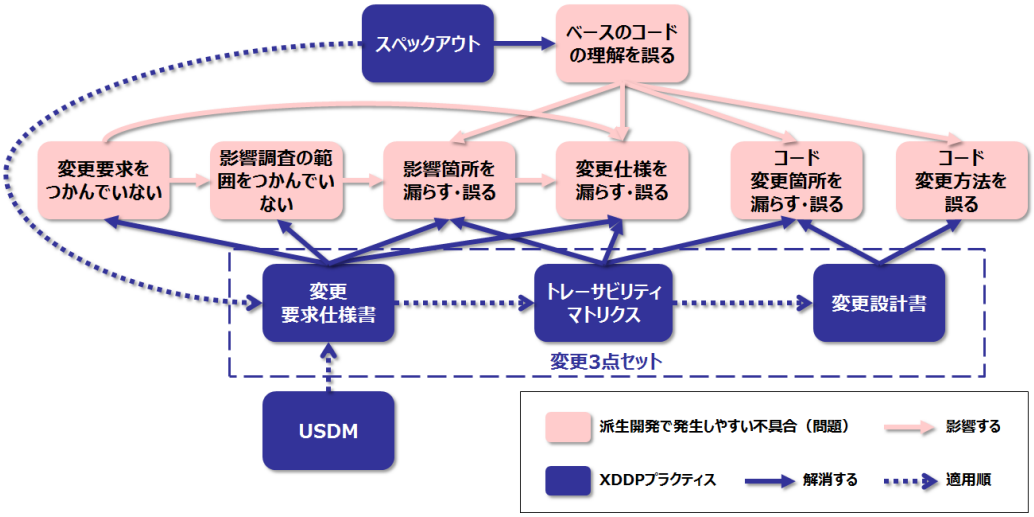

図 2.派生開発で発生しやすい不具合（問題）と XDDP プラクティスの関係

2.4 Philosophy

清水氏は、XDDP を実践する人・組織が持つべきフィロソフィーとして、「肯定眼」を挙げている。肯定眼とは、「他人のプロセスや成果物の中で良いところを積極的に肯定する」(清水 2007b)ことを意味する。例えば、XDDP プラクティスの一つである「ピアレビュー」では、レビュー対象の成果物について、欠陥の指摘を行うだけでなく、肯定意見も出すことを重視している。XDDP を実践する人・組織は「肯定眼」を持つことで、能動的にプロセス改善を行う人・組織へと成長することができる。

表 2 XDDP プラクティス一覧

プラクティス名	概要説明	プラクティス名	概要説明
USDM (Universal Specification Describing Manner)	要求の仕様化技術、及び、要求仕様の記述方法。「要求」「理由」「仕様」を、階層構造で表現する。	要件管理	要求仕様のベースライン設定後の仕様変更プロセス。要求仕様書(USDM形式)と追跡リストで管理する。
PFD (Process Flow Diagram)	プロセスと成果物の連鎖を表現する記法。無駄のない合理的な開発プロセスを設計するための技術。	構成管理	XDDPでは変更制御のプロセスを着実にまわすことで、ベースとなる正規の設計文書の正当性を担保する。
品質要求の仕様化	品質要求に対応したUSDMの拡張版を提供する。品質メトリクス毎に要求仕様を階層表現できるようにする。	品質保証	高品質なソフトウェアを開発するために必要となる品質保証体制とその役割を提唱。プロセスの品質保証には、PFDを用いたプロセス設計技術が必要。
追加機能要求仕様書	追加機能の要求仕様をUSDM形式で表現したドキュメント。	仮説見積り	USDM形式の要求仕様書の要求策定段階で、推定の仕様数と生産性をベースに、開発規模（ステップ数）を見積もること。
スペックアウト	ソースコードを解析して設計資料を作り出す行為。必要な範囲に限定して実施することがポイントとなる。	サイズ見積り	USDM形式の要求仕様書の完成段階で、仕様数と生産性をベースに、開発規模（ステップ数）を見積もること。
変更要求仕様書	変更の要求仕様をUSDM形式で表現したドキュメント。	プロジェクト計画	XDDPプラクティスを活用して、プロジェクト計画書を作成する。
トレーサビリティ・マトリクス (TM)	既存システムに対する変更の影響箇所を確認するためのマトリクス。変更仕様を行、ソースファイルを列で表現し、交点に変更箇所(関数名)を記入する。	スケジュール管理	PFDを用いたプロセス設計技術と要求仕様化技術を手に入れることで、妥当性のあるスケジュールの作成や調整が可能となる。
変更設計書	トレーサビリティ・マトリクスの列単位に、ソースコードの具体的な変更方法を記述したドキュメント。	三段見積りトレーニング	初期見積りの精度を向上させるための見積りトレーニング法。
一斉にコード変更	ソースコードの変更箇所を見つけ次第すぐに変更せず、変更3点セットのレビュー完了後に一斉にコード変更することで、ソースコードの劣化を最小限にする。	Evolutionary Scheduling	PFD上でプロセスを変化させてスケジュールの別案を考える。
DevJTest (Dev Join to Test)	開発者とテスト技術者がプロジェクトの初期段階から情報共有を行い、フロントローディングでバグを減らす取り組み。	新規性管理	個々の開発者が失敗の確率を減らすために、自身が初めて取り組むドメインや要素技術から課題を探し対応する方法。リスク管理としても利用できる。
ピアレビュー	変更3点セットの段階的レビューによって、派生開発の不具合を極限までなくす。また、肯定意見も出すことを推奨する。	リスク管理	要求仕様化のスキームをリスク管理に応用し、「リスク」「要因」「軽減措置」の階層構造でリスクを表現する。
T型マトリクス	トレーサビリティ・マトリクスの行にテスト項目を配置し、開発者とテスト技術者の視点が交差するよう拡張したマトリクス。	レスキュープロセス(火消し)	PFDを活用して、問題プロジェクトを立て直すための手法。
単体テスト	変更設計書の情報を使って、関数レベルでテストを行う。	モジュールの尺度と分割	派生開発のなかでソフトウェアの保守性を維持していくために開発者に求められる設計技術。
結合テスト	テスト設計のインプットとして変更3点セットを活用する。	アーキテクチャ再構築法	オブジェクト指向の考え方を一部取り入れ、短期間で優れたアーキテクチャの確立を目指す方法論。
システムテスト	バグ原因の混入工程が要求の仕様化プロセスにあった場合のプロセス改善を主導する。	リリース	XDDPでアジャイルにリリースする手法。

3. XDDP Patterns

本章では、XDDP にパターン・ランゲージの考え方を応用した XDDP Patterns を提案する。

3.1 What is Pattern Language?

パターン・ランゲージとは、建築家のクリストファー・アレグザンダーによって提唱された（クリストファー・アレグザンダー 1984;アレグザンダー 1993）知識記述の方法で、良いデザインや良い実践の秘訣を共有するための手法である（井庭崇他 2013;井庭他 2016）。

表 3 は、上記の考え方をもとに、パターン・ランゲージの概念を整理したものである。パターン・ランゲージは、パターンとパターン・マップの 2 つの要素から構成される。両者を総称して、「組織パターン」、「スクラムパターン」など、○○パターンと命名されることが多い。

表 3 パターン・ランゲージの構成要素

パターン・ランゲージ	構成要素	説明
○○パターン （○○ Patterns）	パターン （Pattern）	特定の「状況」（Context）において、繰り返し発生する「問題」（Problem）と、「解決策」（Solution）の関係を表したもの。
	パターン・マップ （Pattern Map）	パターンとパターンの関連やパターンの順序の表したもの。 網目のように絡み合ったネットワーク構造を持つ。

パターン・ランゲージは、建築の世界だけでなく、ソフトウェア開発の世界でも活用されている。例えば、アジャイル開発における組織のあり方は「組織パターン」としてまとめられている（図 3）。

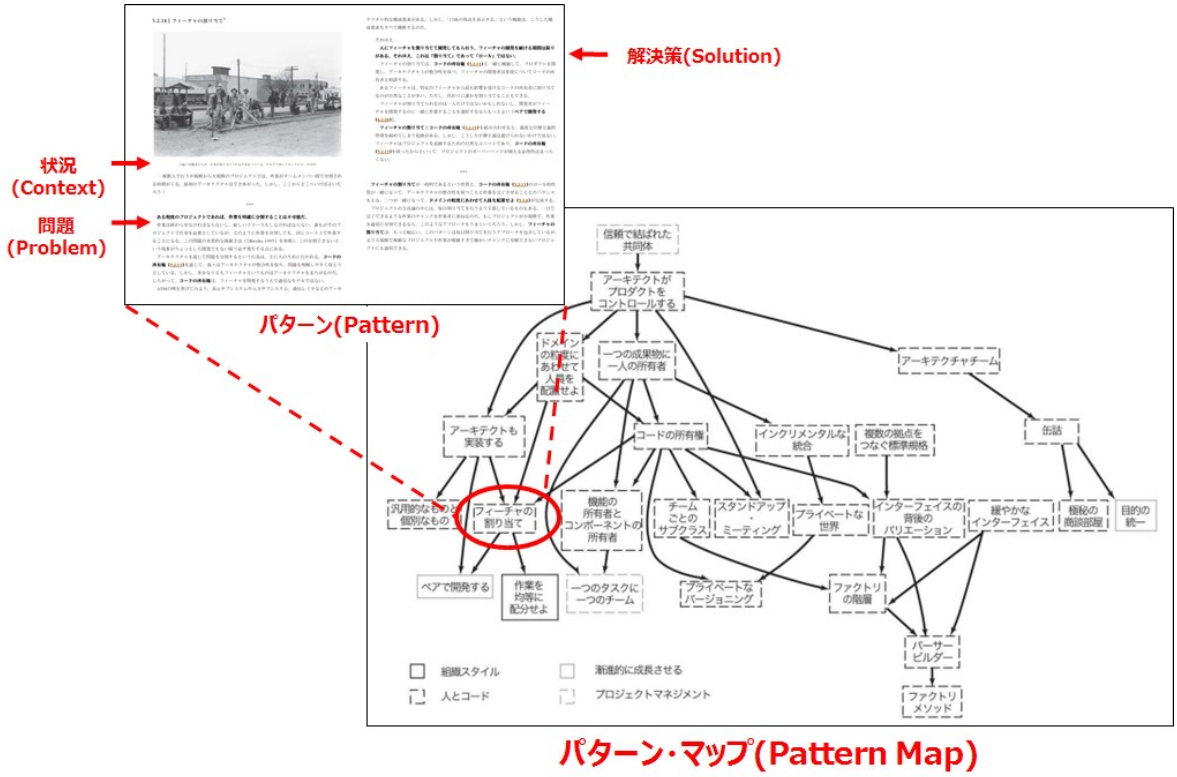

図 3.「組織パターン」におけるパターンとパターン・マップ
出典：James O. Coplien・Neil B.Harrison(2013) p.243, pp.275-276
※赤色の文字・円・点線・矢印は筆者追記

良いデザイン・実践の秘訣をパターン・ランゲージで表現することには、以下のメリットがある。

- 秘訣をパターンという小さい単位で整理しておくことで、つまり、部品化しておくことで、それぞれの現場の状況に合わせて特定のパターンを選択して、再利用することができる。

- パターンどうしの関連をパターン・マップのネットワーク構造で表現することで、複数のパターンを一定の順序で組み合わせるためのルールを知ることができる。

- パターンやパターン・マップを共通言語としてステークホルダー全員が共有することで、ステークホルダー間のコミュニケーションが促進され、全体像を共有することができる。

3.2 Definition of XDDP Patterns

本稿では、XDDP Patterns を「XDDP をパターン・ランゲージの形式で体系化したもの」と定義する。具体的には、個々の XDDP プラクティスをパターンの形式で表現し、XDDP プラクティスの関連や適用順をパターン・マップの形式で表現したものである（表 4）。

表 4 XDDP Patterns の定義

パターン・ランゲージ	構成要素	説明
XDDP パターン (XDDP Patterns)	パターン (Pattern)	個々の XDDP プラクティスを、パターンの形式で表現したもの。
	パターン・マップ (Pattern Map)	XDDP プラクティスの関連や適用順を、パターン・マップの形式で表現したもの。

XDDP Patterns が XDDP 普及課題を解決すると考えた理由は、以下の通りである。

- 個々の XDDP プラクティスを、パターンに凝集し、疎結合化することができる。
  つまり、XDDP プラクティスを部品化することで、選択的・段階的導入を可能とすることができる。

- プラクティスの関連や適用順の表現ができる。
  それにより、各々の開発現場の状況に合わせて必要なプラクティスを選択するためのロードマップとして活用することができる。

- 多様な経験を持つステークホルダー同士が対話をするためのガイドマップを提供する。
  それにより、ステークホルダー全員にとって目指すべき将来像を共有することができる。

3.3 A List of the Patterns

2 章末の表 2 の「XDDP プラクティス一覧」が、XDDP パターン一覧に相当する。

3.4 A Map of the Patterns

以下に示す XDDP プラクティス路線図が、XDDP パターン・マップに相当する（図 4）。

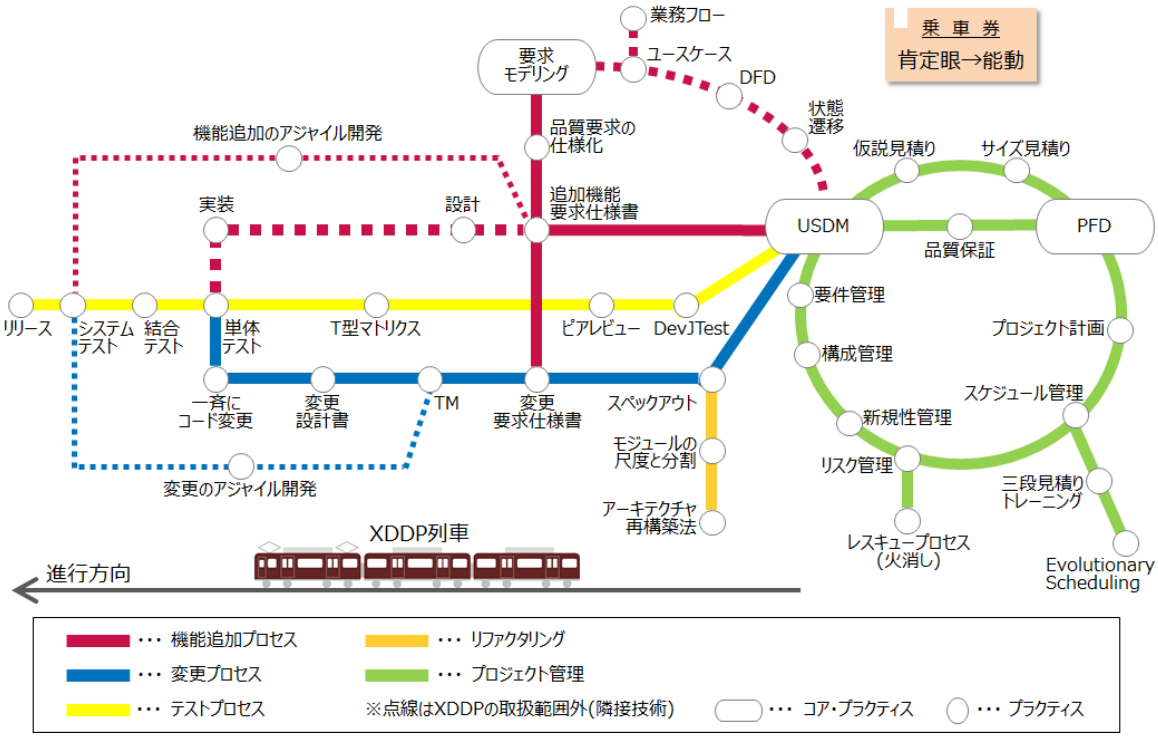

図 4.XDDP プラクティス路線図

筆者は、図 4 を、関西（大阪・神戸）の鉄道路線図をヒントに考案した。鉄道路線図からヒントを得た理由は、鉄道路線図がパターン・マップの網目のように絡み合ったネットワーク構造に似ていたからである。

XDDP プラクティス路線図は、多様なステークホルダーが、以下の①～⑥を直感的、且つ、同時に理解できることを狙って作成した。XDDP プラクティス路線図は、パターンの関連や適用順を表すため、XDDP 導入のガイドマップ、ロードマップとしての役割を果たす。

① プラクティスの関連

プラクティスを駅で表現し、プラクティスの関連は駅と駅を結ぶ線路で表現している。隣接するプラクティスは、現場に同時に導入することでより効果をあげることのできるプラクティスである。

② プラクティスの分類

プラクティスと開発プロセスを対応付けるために、路線という考え方を利用している。同じ路線上のプラクティスは強い関連を持っており、現場の状況に応じてどの開発プロセスの改善に力を注ぐべきかについて、議論を集中することができる。

③ プラクティスの中心

XDDP の中心となるプラクティスは「USDM」である。「USDM」は、どれか一つの開発プロセスに属するのではなく、すべての開発プロセスの基底となっている。そのため、「USDM」を各路線の始発駅が集結するターミナル駅として表現している。関西最大の駅である JR 大阪駅や阪急 大阪梅田駅、阪神 大阪梅田駅などのターミナル駅が集結する大阪・梅田一帯をイメージしている。

④ プラクティスの順序

どのプラクティスをどういう順番で導入するかを、始発駅からの駅の並び順で表現している。ただし、プロジェクト管理は、プロジェクトのモニタリングとコントロールがあり、直線的な構造でなく循環的な構造を持つため、JR 大阪環状線をイメージして表現している。

⑤　フィロソフィー（肯定眼と能動）
　XDDP のフィロソフィーである肯定眼と、肯定眼により引き起こされる能動を、乗車券として象徴化している。XDDP を実践する人・組織は、この乗車券を持っていなければならない。

⑥　導入の主体としての開発メンバー
　開発メンバーを XDDP 列車として表現することで、開発メンバーが主体的に XDDP を導入していくことを象徴化している。大阪―神戸間を、JR・阪急・阪神電車が並走していくように、開発メンバーが機能追加・変更・テストのプロセスを並行して進めていく。

また、鉄道路線図という身近なもので XDDP を表現することにより、XDDP に親しみやすさを感じてもらうことも狙いとしてある。ゆえに、XDDP プラクティス路線図は、多様なステークホルダーがこの図を介して対話できる、現場のコミュニケーション・ツールとして活用できると考えている。

4. Case Study

本章では、筆者が支援していた派生開発現場における、XDDP パターン・マップの活用事例を示す。本事例で取り上げる現場は、大規模基幹システムのリプレースをした直後で、以下のような状況にあった。

- 保守フェーズへ移行した際に要員が大幅に減ったため、開発メンバー一人当たりのシステム担当範囲が増えており、開発の負担が高くなっていた（短納期の制約）。

- 複数の旧システムが統合された結果、新システムの規模が大きくなったため、システム全体を把握している開発メンバーはいなかった（部分理解の制約）。

こうした状況から XDDP 導入が有用であると考え、この現場のプロジェクト・マネージャーに XDDP 導入を提案し、承諾を得た。しかし、開発メンバーは全員 XDDP 未経験者であった。そこで、「特定の知識がなくてもプラクティスの全体像が理解可能なガイドマップ」を示して理解を促した上で、「現場に負担のない、最低限のプラクティスを導入するロードマップ」を作成することが必要と考えた。
以上より、3 章で述べた XDDP のパターン・マップである XDDP プラクティス路線図を、この現場に適用することとした。

4.1 Application of XDDP Pattern Map

筆者は、図 5 に示すとおり、①現状分析、②XDDP とのマッピング、③導入プラクティスの決定、④開始プラクティスの決定の手順で、この現場の XDDP 導入ロードマップを作成した。

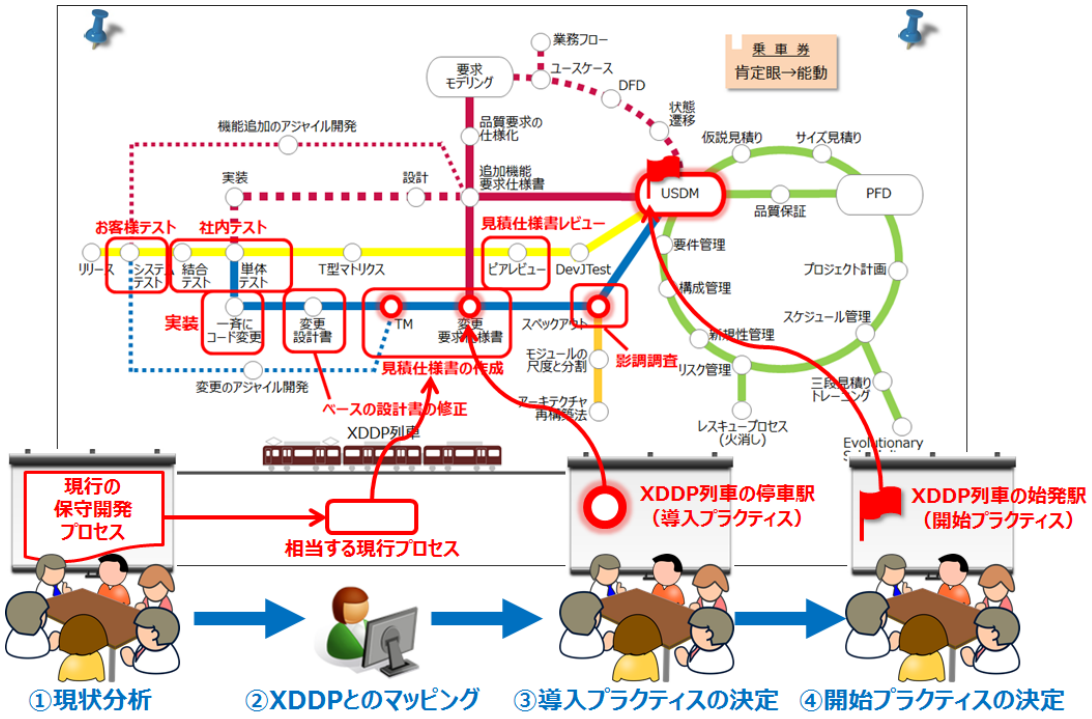

図 5.XDDP 導入ロードマップ作成の手順

以下、それぞれのステップについて述べる。

① 現状分析

開発メンバーへのヒアリングやプロジェクト資料の確認を通して、現行の開発プロセスと成果物の連鎖の見える化を行った。見える化には、XDDP プラクティスの一つである PFD（プロセス・フロー・ダイアグラム）を活用した。

② XDDP とのマッピング

①で分析した結果と XDDP プラクティス路線図とのマッピングを行った。マッピング結果は、XDDP プラクティス路線図に「相当する現行プロセス」として追記した。
これにより、現行の開発プロセス（As-Is）と XDDP が求めるあるべき開発プロセス（To-Be）の差分を、開発メンバーが直感的に理解できるようにした。

当ステップで、XDDP プラクティス路線図を「多様なステークホルダーが理解可能なガイドマップ」として活用した。

③ 導入プラクティスの決定

筆者がファシリテーターとなり、②の XDDP プラクティス路線図を開発メンバーに見せながら、どのプロセスを改善する必要があるか開発メンバーにディスカッションしてもらった。この現場では、変更プロセスの上流工程を改善すべきという結論に至った。そして、プラクティス導入候補として「スペックアウト」、「変更要求仕様」、「トレーサビリティ・マトリクス」があがった。

当ステップで、XDDP プラクティス路線図を「選択的導入のロードマップ」として活用した。

④ 開始プラクティスの決定

③で選択したプラクティスを導入するには、「USDM」を最初に理解する必要があることを XDDP プラクティス路線図が指し示している。そのため、開発メンバーの間で、「USDM」からスタートすることがスムースに合意形成できた。

当ステップで、XDDP プラクティス路線図を「段階的導入のロードマップ」として活用した。

4.2 Results and Discussion

本事例での XDDP パターン・マップの評価、及び、今後の課題について、以下に述べる。

評価

① ガイドマップ、ロードマップしての評価

XDDP プラクティス路線図が持つ理解のしやすさにより、XDDP の全体像を現場に容易に伝えることができた。本事例の開発メンバーは全員 XDDP 未経験者であったが、XDDP プラクティス路線図を活用することで、プロセスを改善するために、どのプラクティスを導入すべきかについてのロードマップを描くことができた。

② コミュニケーション・ツールとしての評価

現場のディスカッションでは、鉄道路線図の親しみやすさから、終始リラックスした雰囲気が醸成され、改善すべきプロセスについて活発に意見を引き出せた。例えば、「この辺りの区間で列車がよくバックする」といった表現で、手戻りが多いプロセスについて話してくれたメンバーがいた。XDDP 列車に例えて話すことで、問題について話しやすくなった。

今後の課題

この現場では、開発がうまくいくよう独自の工夫を行っている人がいることがわかった。他の開発現場も同様に、その現場のコンテキストに合った独自の工夫をしていると思われる。よって、それらの工夫を、XDDP Patterns の新しいパターンとして、つまり、「新駅」として追加できるステップを追加し、それぞれの現場独自のパターン・ランゲージとして拡張できるようにしていきたいと考えている。

5. The Patterns

本章では、XDDP パターンのうち、「USDM」、「スペックアウト」、「一斉にコード変更」について記述する。ただし、パターン記述としては、まだ卵の段階であり、今後も継続的に改善していく必要がある。また、上記以外のパターンの記述についても、今後、拡充していきたい。

XDDP パターンは、アレグザンドリアン形式で記述する。アレグザンドリアン形式を採用したのは、この形式は、俳句のような美しいリズムや間があり、日本語のパターン記述に大変マッチしていると考えているからである。

USDM

要求の仕様化技術、及び、要求仕様の記述方法

すべての道は USDM に通ず。

...開発者は依頼者から要求仕様を聞き出そうとしている。

* * *

依頼内容を受けて“そのまま”開発すると、本来必要な仕様が漏れていたり、不適切な仕様で開発したりしてしまうことがある。

派生開発の場合、実際に稼働しているシステムが存在するので、依頼者はシステムの具体的な追加・変更内容を伝えてくることが多く、その背景にある依頼者がシステムで実現したいことを伝えてくることは少ない。そのため、考慮すべき影響範囲、つまり、仕様化すべき範囲が曖昧である。

曖昧な依頼内容のままではなく、それらを構造化して表現することが求められる。

Therefore:

依頼内容を分解し、「要求」、「仕様」、「理由」に再構成する。要求仕様は「要求」と「仕様」とに分け、階層構造で表現する。「要求」には依頼者がシステムで実現したい“こと”を表現し、仕様化の範囲を示す。「仕様」にはソースコードに変換可能な“もの”を表現し、要求を満たすシステムの振る舞いや処理などを示す。**また、「要求」には必ずセットで「理由」がある。**「理由」を明確にし、「要求」の導出や洗練、妥当性確認の拠り所にする。

＊＊＊

依頼内容に仕様しかない場合は、仕様から要求・理由へとリバースし（逆展開し）、突きとめた要求から下位要求や仕様へと展開する流れでやるとよい。

スペックアウト

ソースコードを解析して設計資料を作り出す行為

作った人はもういない。

...新規開発時に後の派生開発のときに有効な設計資料は作られていないため、ソースコード以外に仕様を理解する手がかりがない。そのため、ソースコードを読んで既存システムの仕様を理解しようとしている。

＊＊＊

ソースコードの理解を誤ると、以下のミスの原因となる。

- **影響箇所の漏れ・誤り**
- **変更仕様の漏れ・誤り**
- **コード変更箇所の漏れ・誤り**
- **コード変更方法の誤り**

一般に、他人の書いたソースコードから仕様や設計の意図を読み取ることは難しい。
また、歴史の長いシステムであると、ソースコードを書いた当人がおらず、質問できないことも多い。

Therefore:

ソースコードを読んで理解したことを図や表などで表現する。
ソースコードを別の表現に変換することで、理解状況（「わかった」ということ）を可視化できる。
理解状況を可視化すれば、第三者が確認することができ、関係者の理解のずれに気づくことができる。

＊＊＊

図や表などを使って表現できるよう、PAD フロー、デシジョンテーブル、UML などの表現技術を習得する。また、可視化した資料は、今後の派生開発で活用できるよう、ベース・ドキュメントの更新に取り込む。

一斉にコード変更

足並みを揃えてソースコードを変更する

『源平合図屏風』「一ノ谷」
単騎で敵陣に突入してはいけない！

...***変更設計書***を作成し、ソースコードの変更方法のレビューも完了した。いざ実装だ！

* * *

一度ソースコードを変更してしまうと、後から良い変更方法が見つかっても、開発者は変えたがらない。

> 多くの人は、ソースコードを書いたり変更したりすると、そのソースコードを“守って”しまいます。そこが「実装」と「設計」とで違うところでしょう。「設計」の場合は、何度もやり直したり書き直したりすることにはほとんど抵抗はありません。実際にデータ構造の図を何度も書き直しますし、クラス図やモジュールの構造図も適切な尺度と照らしながら書き直すことには抵抗はありません。しかし、いったんソースコードを書いたあとで変更が必要だとか、あの変更は意味がなかったとなれば、“せっかく書いたのに”という言葉がでてきて、いまからそれを書き直すことに抵抗します。現状の変更方法が最適な方法ではないとわかっても、変更しなくても済む方法はないかと探します。こうして不正な変更がまかり通ってしまうのです。（清水 2007a）

Therefore:

ソースコードの変更は、変更 3 点セットのレビュー完了後に一斉に実施すること。
複数の開発者がいる場合は全員分のレビュー完了後（整合性確認後）に実施する。

* * *

XDDP を導入すると、導入前と比べて、ソースコードの変更は短期間で終る。
なぜなら XDDP 変更 3 点セットでやる事がクリアになっており、この段階ではもう悩まないためだ。

REFERENCES

清水吉男(2007a)『「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
清水吉男(2007b)「２１世紀の品質保証体制の構想」
<https://affordd.jp/koha_hp/Else/Head0701.html >(2020/1/22 アクセス)
清水吉男(2010)『【改訂第 2 版】[入門＋実践]要求を仕様化する技術・表現する技術―仕様が書けていますか？』、技術評論社
クリストファー・アレグザンダー(平田翰那訳)(1984)『パタン・ランゲージ―環境設計の手引』、鹿島出版会
クリストファー・アレグザンダー(平田翰那訳)(1993)『時を超えた建設の道』、鹿島出版会
井庭崇他(2013)『パターン・ランゲージ―創造的な未来をつくるための言語』、慶應義塾大学出版会
井庭崇他(2016)『プロジェクト・デザイン・パターン―企画・プロデュース・新規事業に携わる人のための企画のコツ 32』、翔泳社
James O. Coplien・Neil B.Harrison(和智右桂訳)(2013)『組織パターン―チームの成長によりアジャイルソフトウェア開発の変革を促す』、翔泳社
長坂一郎(2015)『クリストファー・アレグザンダーの思考の軌跡―デザイン行為の意味を問う』、彰国社
川口典子(2017)「XDDP 普及のための XDDP プラクティス路線図の提案」、第 55 回 IBM ユーザー論文（IBM Japan Users Association）