

AFFORDD活動紹介① AFFORDD研究会の紹介

2024/5/24 派生開発カンファレンス2024

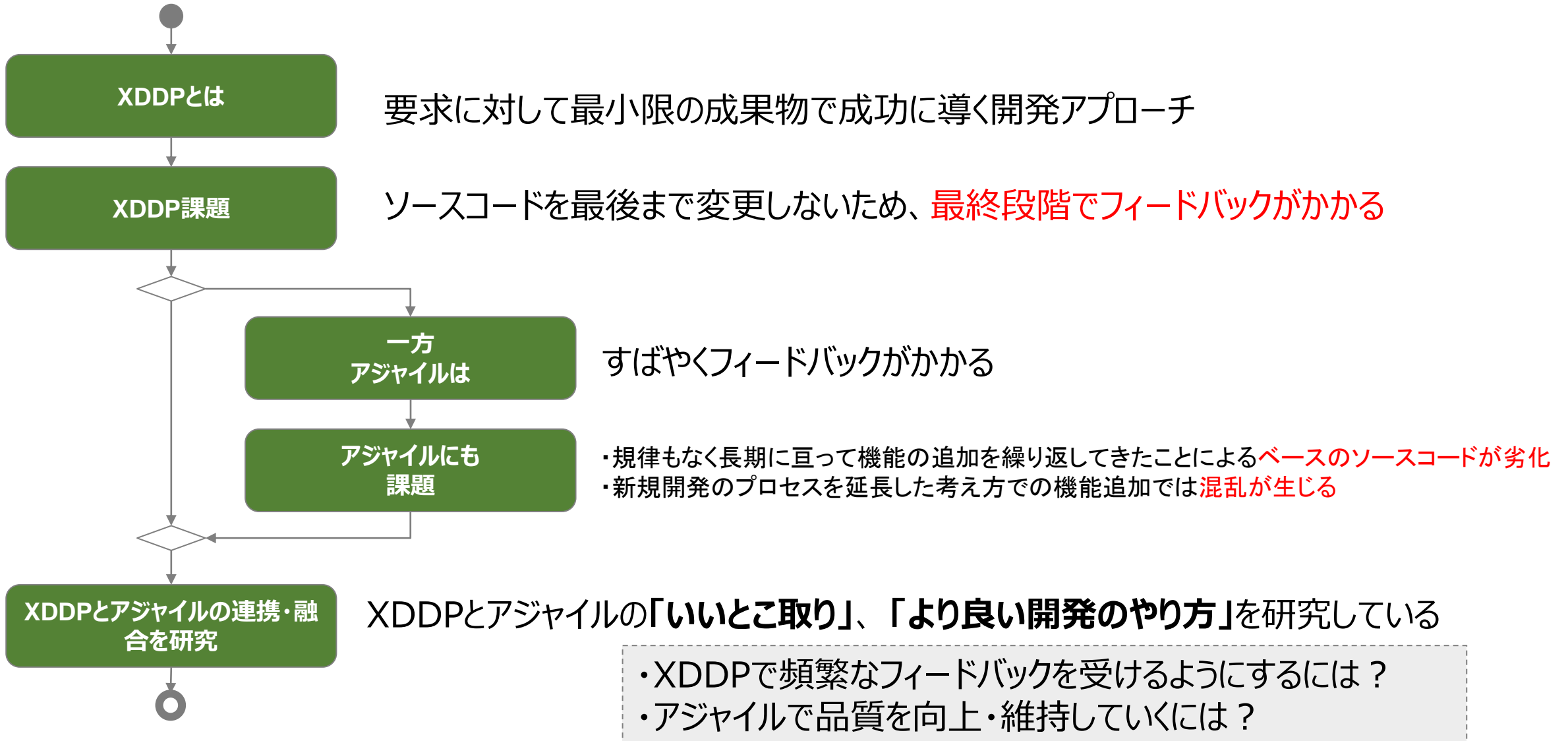
- AFFORDDは、「USDM」「XDDP」「PFD」の技術を学び獲得するだけでなく、派生開発にまつわる様々なテーマを研究し、深める活動をしています。
- AFFORDD会員は各自興味のある研究会に所属し、メールやDiscord、対面での議論を通して活動成果をまとめています。
- ここでは、研究会の最近の活動内容について紹介します。

T6研究会

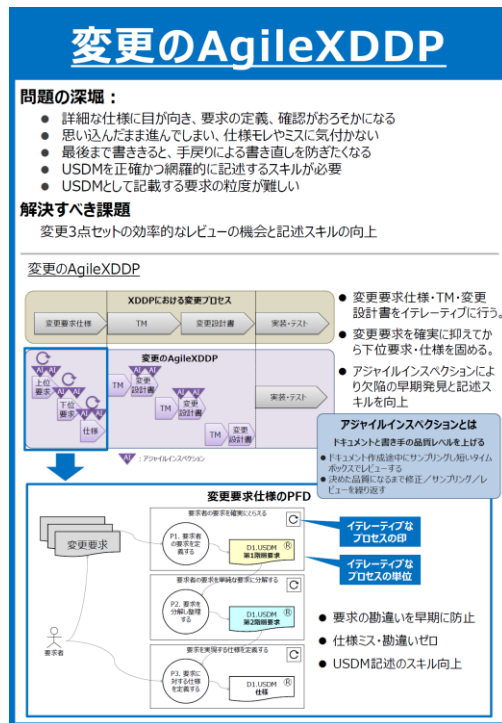
Agile開発との連携

- Agile開発に対する課題
 - 規律もなく長期に亘って機能の追加を繰り返してきたことによるベースのソースコードが劣化
 - 開発に携わるメンバーも交替
 - ベースのアーキテクチャの設計が最初から不安定
 - 新規開発のプロセスを延長した考え方で機能追加では混乱が生じる
- もともと「XDDP」自身が、要求に対して必要最小限の規律を求めただけの「機敏な」開発アプローチ
- 「機能追加」のプロセスを「Agile」な開発技法で対応し、追加機能の受け入れを含む「変更」のプロセス全体を「XDDP」で対応することが可能ではないか？

「Agile」な開発技法と「XDDP」を繋いで効果をあげるには？



- Agileの課題とXDDPの課題に対して、双方の課題を解決する「AgileXDDP」を考案(2015)
- Agile開発には、派生開発的な問題にXDDPのプロセスを適用
- XDDP開発には、Agileプラクティスを適用



AgileXDDP

AgileXDDPとは？

2つのAgileXDDPから成り立つ、Agileの課題とXDDPの課題の両方を解決する手法です。

変更の AgileXDDP

XDDPの課題を解決する「変更」のための手法

機能追加の AgileXDDP

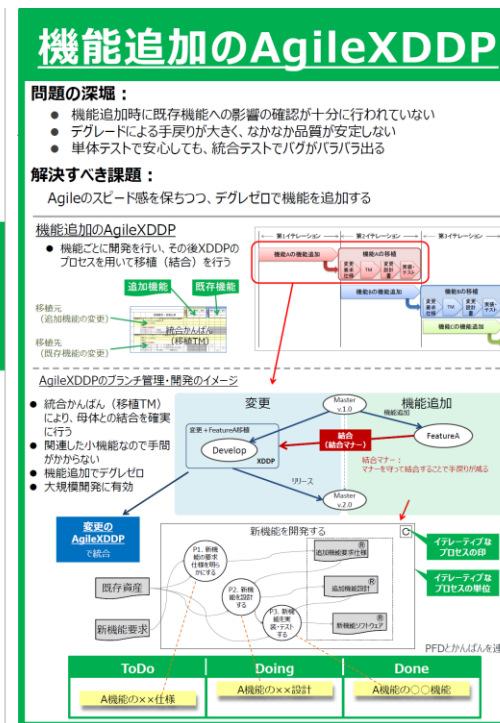
Agile開発の課題を解決する「機能追加」のための手法

XDDPの問題

- 変更3点セットを進めても、適切なレビュー機会がないと、モヤミスが発生する。
- 変更3点セットの各成果物を書ききってからレビューを行うと手戻りが発生しやすい。
- 変更要求仕様書に用いられるUSDMを、正確かつ網羅的に記述することが難しい

Agile開発の問題

- イテレーションを繰り返す、ソフトの規模が増大すると、デグレードの問題が多発する。
- 既に動いているソフトウェアへ、Agile開発を適応し、機能を追加するのは難しい。特に、仕様やテストケースが失われている場合、デグレードを防ぐために多大な労力が必要になる。



- ScrumとXDDPの問題を解決する「スクラムX」(2017)
- ビジネス重心のScrumにはソースコード劣化の予防は規定なし
- XDDPを導入している派生開発でも、高速リリースの要求高

スクラムX

Scrumの特徴・課題	XDDPの特徴・課題
■特徴 <ul style="list-style-type: none"> アジャイル開発は動くソフトウェアにインクリメンタルに追加および変更開発を繰り返していく(差分開発) 仕様の表現や詰め方、設計のやり方についてはScrumでは規定していない ■課題 <ul style="list-style-type: none"> 開発チームの行うプロセスが明確でないため、調べた内容を残す方法がなくなり、最適な仕様と設計の吟味ができなくなり、修正前の影響範囲などのレビューが難しく、手戻りが発生する 	■特徴 <ul style="list-style-type: none"> 変更は既存資産に対する「差分」で進め、既存資産の不明な部分はスペックアウトで明らかにする 変更3点セット「変更要求仕様書」「TM(Traceability Matrix)」「変更設計書」により、具体的な変更方法を早期に確認する 変更方法確認後、一気にソースコードを変更することで結合後の混入がない ■課題 <ul style="list-style-type: none"> XDDPを適用すべき状況において、プロダクト要求による段階的なリリースや、要求の優先順位の変更が求められるような開発でその都度現場で考える必要がある

ScrumとXDDPの問題を解決する「スクラムX」

- Scrumはビジネス重心であり、既存システム(ソースコード)の理解が難しい場合はうまく進まない
- 派生開発の要求にも優先順位があり、優先順位が高いものは早くリリースしたい
- ScrumとXDDPの融合により、上記課題を解決
 - ⇒ スクラムXの提案 (X: eXtreme, eXpend, eXtension, Xddp)
- スクラムXの特徴**
 - 最初に変更に対する既存システムの不明点を明確にし、プロジェクトの要求の優先順位を、ビジネスの側面だけでなく、システムへの影響を加味して進める
 - 役割: Scrumのプロダクトオーナー、スクラムマスターに「スカウター」を追加
 - プロセス: ファーストダウン・タッチダウンをクォーター単位で実施
 - **ファーストダウン**: 変更の影響範囲を見極める
 - **タッチダウン**: ファーストダウンの範囲で変更設計・ソースコードを変更
 - **クォーター**: ファーストダウン・タッチダウンを1つの単位としたもの。リリース、デプロイの単位

スクラムXの効果

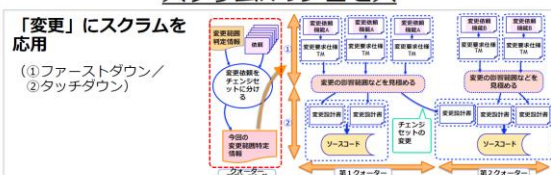
- 実施するプロセスが明確になり、レビューが行えるようになり、抜け漏れなどがなくなる
- XDDPの課題であった計画的な段階リリースが可能になる

スクラムXの役割

プロダクトオーナー	■ プロダクトについて何をどの順番で作るのかについて責任を持つ
スクラムマスター	■ プロジェクトへのスクラムの導入、エンジニアリングプロセスの観点で開発チームをマネジメントする
スカウター	■ システムの構造を理解し、変更要求による影響範囲を見極める(既存設計を偵察、すなわち、スカウティングする役割から、スカウターと呼ぶ) ■ 内部の作りによって要求の優先順位の見直しをプロダクトオーナーに提案する(戦術参謀)
開発チーム	■ 開発チームはプロダクトバックログアイテムを開発し、リリースすることに責任を持つ ■ バックログをリファインメントする (USDMで要求を仕様化する)

スクラムXのプロセス

「変更」にスクラムを応用
(①ファーストダウン/②タッチダウン)



この図は、変更要求の発生からリリースまでのプロセスを詳細に示している。主要なステップは以下の通りである:

- 変更要求の発生**: 変更要求の発生、変更要求の登録、変更要求の優先順位を決定する。
- ファーストダウン**: 変更要求の優先順位を決定する、変更要求の範囲を見極める、変更要求の範囲を決定する。
- タッチダウン**: 変更設計の作成、ソースコードの変更、ソースコードのレビュー、ソースコードのテスト。
- クォーター**: ファーストダウン・タッチダウンを1つの単位としたもの。リリース、デプロイの単位。

- その他にもAgileプラクティスとXDDPの手法の融合について議論と実践を行っています
 - XDDPの3点セットをAgile開発の中にとりこみ要求獲得、要求分析フェーズでの課題にUSDMを用いる
 - PFDからプロダクトバックログへの落とし込み
- Agile開発は日本の開発現場には浸透してきつつありますが、派生開発の現場で活用するには、課題はまだまだたくさんありXDDPも時代の変化によって、高速化・俊敏性が求められています
- XDDPの利点をAgile開発にとりこみ、Agileの俊敏さをXDDPに適用していく、研究をこれからも行っていきます

T13研究会

「USDM」のリスク管理への応用

- USDM (Universal Specification Describing Mannerの略) とは、要求を適切に表現することで、要求から仕様を漏れなく引き出すこと、関係者が内容を特定できることを狙いとした、要求と仕様の表記法です。
- USDMは、要求と仕様を階層化された構成で記述するための『要求仕様書（様式・フォーマット）』と、要求と仕様を適切に表現するための『記述マナー（Describing Manner）』から成り立っています。

USDMの要求仕様書フォーマット（最小構成）

カテゴリ名 (記号)	要求	(要求ID)	
		理由	
		説明	
	<仕様グループ>		
	□□□□	(仕様番号)	
	□□□□	(仕様番号)	
	□□□□	(仕様番号)	
	<仕様グループ>		
	□□□□	(仕様番号)	
	□□□□	(仕様番号)	
	□□□□	(仕様番号)	
	<仕様グループ>		
	□□□□	(仕様番号)	
	□□□□	(仕様番号)	
	□□□□	(仕様番号)	

要求の配下に仕様を配置する階層化された構成

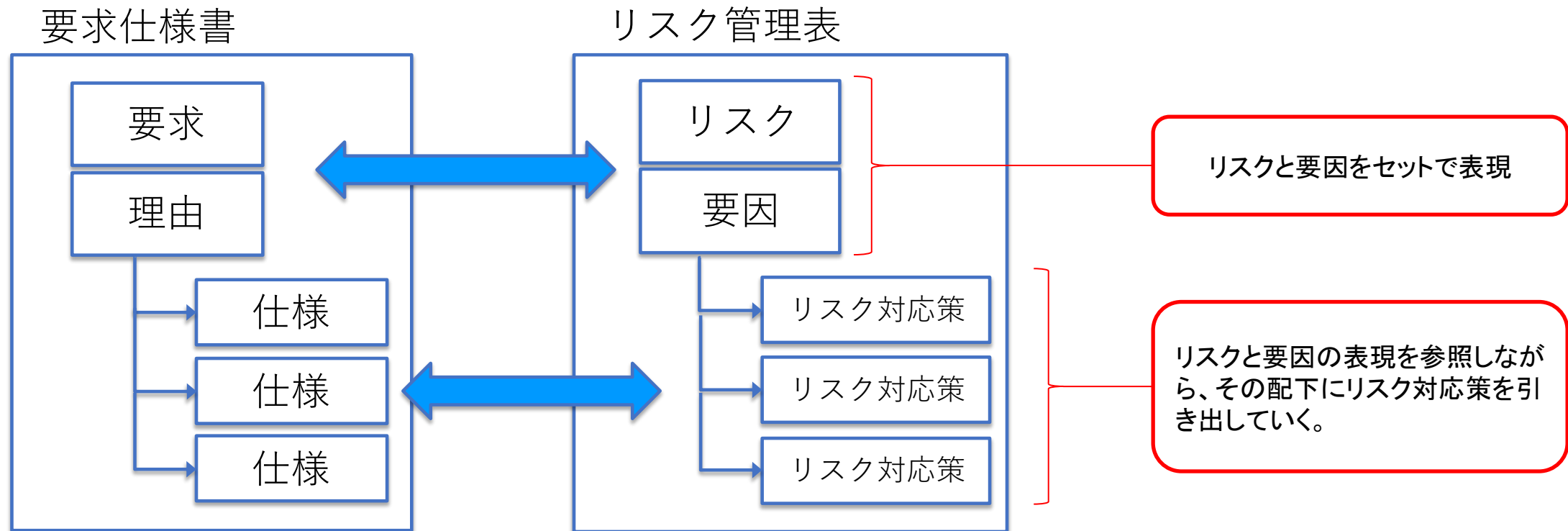
要求と、その要求が出てきた理由(背景)をセットで表現

仕様をグループ化することで仕様漏れに気づきやすくする



参考資料 【T2研究活動成果】 USDMを簡単に理解してもらうための小冊子
<https://affordd.jp/libraries/affordd-t2-usdmtext/>

- USDMの特徴である、要求と理由をセットで表現し、そこから仕様を引き出す構造が、リスクと要因を識別して、そこからリスク対応策を引き出す構造と類似しています。
- 要求と仕様を階層化された構成で記述する形をリスク管理表にも応用できます。



- ・ Xリスク管理表のフォーマット（全体像）
USDMの階層構造を、リスク管理に使う要素に対応させた構造にカスタマイズ。

2 リスク分析（評価）記述エリア

リスク	(リスクID)	コントローラとPCの相性問題発生により、システムテスト開始が遅れるリスク（可能性）		発生確率・影響度 (before)		損失額	
		定性的 (5段階)	定量的 (%)	総額	期待損失	定性的 (5段階)	定量的 (%)
1 リスク記述エリア	状態・状況	【状態】 今回採用予定のコントローラボードは新規に開発されたものである。 【状態】 今回採用予定のコントローラボードは今回使用するサーバーPCで使用実績がない。					
	原因	C1	C2				
	不確定要素	今回採用予定のコントローラボードと今回使用するサーバーPCとの間にデバイス接続上の不整合が存在する可能性があるが、不整合が何件くらいあるのか、不整合の深刻さはどの程度なのか、事前の予測は難しく、やってみないと分からない。					
	事象	E1	E2	3	50		
	影響	I1	I2	5	90	¥3,500,000	¥1,575,000

発生確率・影響度 (After)		損失額
定性的 (5段階)	定量的 (%)	期待損失
2	30	
3	50	¥25,000

- ・ 5つの記入エリアは、リスクマネジメントプロセスの段階に応じて記入。
- ・ 1の「リスク記述エリア」を始めとして、それぞれの記入エリアは次のプロセスのインプットになっている構造。

3 リスク対応策記述エリア

<<リスク対応策>>						
		対応策実施者	対応策実施時期	対応するリスク要素	対応策の分類	予想効果 (5段階)
<<FMEA実施による不整合箇所の特定、障害の軽減化>>						
000	1-1	開発者	設計フェーズ開始までに	要因	軽減	3
000	1-2	プロジェクト推進者	設計フェーズ開始までに	事象	軽減	2
<<知見者側面による障害発生頻度の軽減>>						
000	2-1	開発者	結合テスト中	影響	軽減	2

5 対応策実施後リスク分析（評価）記述エリア

4 コンティンジェンシープラン記述エリア

<<コンティンジェンシープラン（CP）>>			CP実施者	対応するリスク対応策
<<テスト追加投入による連携加速>>				
000	C-1	<トリガ> 結合テスト期間の2/3経過時点（〇月〇日）で、未解決のデバイス接続不良問題が10件以上あった場合。	プロジェクト推進者	1-1
000	<->	<トリガ> 特になし。	開発者	

・ Xリスク管理表のフォーマット (リスク記述エリア)

リスク		(リスクID :)		コントローラとPCの相性問題発生により、システムテスト開始が遅れるリスク (可能性)	
リスク要素	要因	状態・状況	C1	【状態】 今回採用予定のコントローラボードは新規に開発されたものである。 【状況】 今回採用予定のコントローラボードは今回使用するサーバPCで使用実績がない。	C2
		不確定要素	C1	今回採用予定のコントローラボードと今回使用するサーバPCとの間にデバイス接続上の不整合が存在する可能性があるが、不整合が何件ぐらいあるのか、不整合の深刻さはどの程度なのか、事前の予測は難しく、やってみないと分からない。	
	事象	E1	E1	結合テスト工程において障害が多発する。	E2
	影響	I1	I1	障害対応の工数、結合テストやり直しの工数が増加し、システムテストの開始が遅れる。	I2
				【要因の説明 (理由・背景)】 【事象の説明 (理由・背景)】 【影響の説明 (理由・背景)】 過去プロジェクトの事例から、コントローラとPCとの相性問題に起因する障害解決には、1件あたり平均3.5日かかっている。障害発生件数見込みを6件と仮定し、3週間程度 (3.5日×6件) の遅れと見込む。	

・ リスクの要因から事象、影響までを1文で表現することで、要因から影響までの関係性、因果関係を意識させる狙い。

・ 要因、事象、影響それぞれ個別に記述。
 ・ 記載欄を分けているのは、何を、要因、事象、影響として捉えているのか、関係者が特定 (Specify) しやすくするため。

- ・ 近日中に小冊子を公開予定。

「Xリスク管理表ガイドライン～リスクを表現する方法、対応策を引き出す方法～」

【主な内容】

- ・ リスクの基本的概念
- ・ Xリスク管理表のフォーマットと記入項目・記入方法
- ・ リスクの要因、事象、影響の引き出し方法
- ・ リスク対応策の引き出し方法

T19研究会

派生開発におけるスペックアウトの仕方

- 派生開発におけるスペックアウトの仕方と、その対応性や表現の方法、スペックアウト時の注意点などについて、掘り下げることがを目的に活動している研究会です。

【派生開発におけるスペックアウトとは】

ソースコードを読んで理解しながら変更箇所や影響箇所を特定し、設計書の一部を生成すること

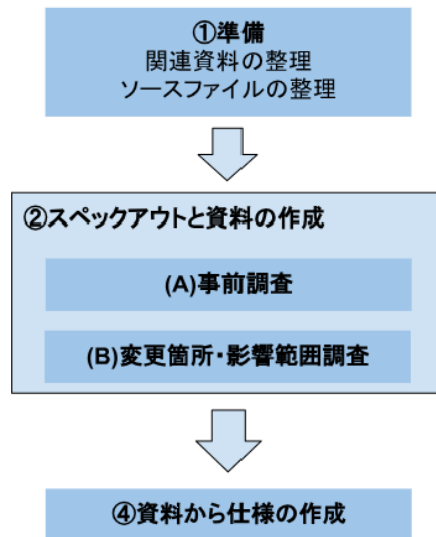


各種資料はAFFORDD公式ホームページから入手可能です。

年度	活動内容
2016年	派生開発カンファレンス2016でポスター展示: 「既存のソースコード10万行を今度の製品に使おうと思っているが、使えるかどうか1週間で判断してほしい」
2017年	派生開発カンファレンス2017でポスター展示: 「何っ?!状態遷移設計を知らない?」
2018年	状態遷移のスペックアウトの実施 派生開発カンファレンス2018でポスター展示: 「コード→状態遷移表→状態遷移図→仕様までのスペックアウトの実際」
2019年	データ構造のスペックアウトの実施 ET & IoT Technology 2019 AFFORDDスペシャルセッションにて発表: 「スペックアウトによる設計仕様の理解」
2020年	データ構造のスペックアウトの実施
2021年	スペックアウトワークショップの準備
2022年	派生開発カンファレンス2022で スペックアウトワークショップ 開催 勉強会
2023年	スペックアウトの小冊子作成

- 研究会での活動で得た知識や経験を元に、スペックアウトの意義、やり方、種別と目的、こういったアウトプットを残すのか、といった内容を小冊子にまとめ中。

- スペックアウトのやり方
スペックアウトの全体の流れとしては、①準備、②スペックアウト、③変更要求仕様(USDM)へ反映の手順で行います。



- スペックアウト
 - スペックアウトの種類
スペックアウトの種類により残す成果物が異なります。以下は種類と成果物の対応を示した例ですが、目的や知りたいことに応じて調べ方や成果物は異なるため、まずは今回の目的でこういったスペックアウトが必要になるのか考えるところからは始めてみると良いかもしれません。

スペックアウトの段階	スペックアウトの種類	知りたい情報	成果物（例）
事前調査	処理構造	<ul style="list-style-type: none"> 関数の呼び出し構造 処理の流れ 	<ul style="list-style-type: none"> 構造図 クラス図 シーケンス図
	状態遷移	<ul style="list-style-type: none"> どんな状態があるのか どんなイベントがあるのか イベントと状態の関係性（移り変わりのルール） 	<ul style="list-style-type: none"> 状態遷移表 状態遷移図
	データ参照関係	<ul style="list-style-type: none"> データの構造/要素 どこからアクセスされているか どのデータと関連しているか 	<ul style="list-style-type: none"> データ構造図 ER図 呼び出し関係図
変更箇所・影響範囲調査	—	<ul style="list-style-type: none"> 変更箇所 影響箇所 	<ul style="list-style-type: none"> USDM

T23研究会

AIと派生開発技術の連携

活動のキッカケ

効率化

- AIで楽できるかな？
- 要求仕様やテストドキュメントでAIをどう使えば効率化できる？
- 中身はあるが、うまく表現できない→日本語を直してくれる

対話

- 清水さんBot
- FAQの自動化
- 思考を鍛えたい

AI搭載製品

- AIを使ったシステムに対する要求仕様ってどう書くの？
- 品質はどこまで保証するの？

活動のキッカケ→現状

効率化

- AIで楽できるかな？
- 要求仕様やテストドキュメントでAIをどう使えば効率化できる？
- 中身はあるが、うまく表現できない→日本語を直してくれる

みんなが楽できる

プロンプトガイドラインを思考中

対話

- 清水さんBot
- FAQの自動化
- 思考を鍛えたい

Fine-tuningやRAGは環境や維持の

面で**難しい(様子見)**

AI搭載製品

- AIを使ったシステムに対する要求仕様ってどう書くの？
- 品質はどこまで保証するの？

ここまで活動に**至っていない**

- ステークホルダの目的に沿った対話が大事
- 入力の文章、言葉が境界から外れるとハルシネーションを起こす
- ベクトルの情報を狙い通り構築しないと、誤った回答を返す

活動のキッカケ→現状→例

みんなが楽しめる
プロンプトガイドラインを思考中

AI検討マトリクス

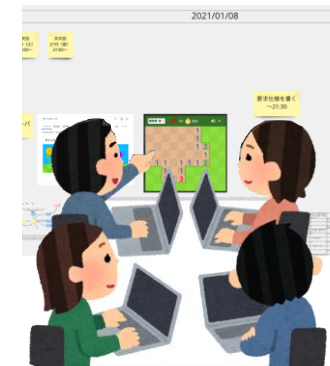
AFFORDDの技術スコープ		AI技術カテゴリ(自然言語モデル)					
技術カテゴリ	アプローチ	文章生成	要約	機械翻訳	感情分析	分類	対話
要求エンジニアリング	USDM系 - USDM - アジャイルUSDM						
	共通 - 要求収集						
	共通 - 要求分析				○		
	共通 - 要求記述	○	-				
	共通 - 要求検証	プロダクトの利用ユーザ/ユースケースの作成・検証				要求の記述のボジネガ判定	品質特性に分類ができるか?
共通 - 要求マネジメント							
プロセス設計	PFD						
派生開発プロセス	XDDP						
	スペックアウト						
	スクラムX						
	共通 - 要件定義						
	共通 - 詳細設計						
	共通 - コーディング						
	共通 - レビュー						
共通 - テスト							



プロンプト例
.....
回答
.....

その他研究会活動紹介

研究会	紹介
T01：障壁の克服方法	派生開発を進める上でXDDPを導入することが効果的ですが、自身の職場に提案しても「今までのやり方で何が悪いの？」と 変化への抵抗 が見られます。いま問題ないと考えている人に対して気づきを促し、やる気になってもらう方法として「 プレモータム（事前検死） 」という思考法を広げることが大事なのではと考え、研究会で自ら実践中です。
T02：「USDM・XDDP」の入門	要求仕様を表現する記法であるUSDMは、単体でも要求を深く理解し手戻りを防ぐ効果があり、XDDPのキーとなる技術の1つでもあります。しかし、USDMの習得には手数をこなすことと、書いたものを評価することが必要です。私達は初心者を中心に USDMの実践とレビューの場を用意 しています。
T08：大規模システムへの効果的対応	XDDPを大規模システムに適用する場合、変更箇所が多くなることから適切な管理方法が必要になります。私達は階層を3階層（システム/サブシステム/コンポーネント）に分け、モデルベースシステムズエンジニアリング（MBSE）の技法も取り入れながら、 大規模システムに向けたXDDPの導入ガイド を作成中です。



研究会	紹介文
T21：「PFD」によるプロセス設計	プロセスを表現し設計するツールであるPFDは、状況の変化に対応するために有用なツールです。2021年にPFDを使いこなすための書籍も出版しました。いま、私達はスケジュールや見積に関する手法を清水さんに提供いただいた資料をもとに学び、「約束」を実現しつづけるためにどうプロセスを設計し、改善に繋げるかを検討しています。
T22：失敗事例	XDDPを導入した成功事例は多くありますが、一方で導入に失敗した事例もあります。導入失敗時には共通の要因があり、これを「病気」のアナロジーで整理し「治療の処方せん」「予防の処方せん」としてまとめ、小冊子として公表しています。さらに、私達が整理した「病気」の整理はリスクマネジメントの各要素と対応していることが判明したため、T13「リスク」研究会とコラボして探求中です。



Amazon書籍で“PFD”で検索！

治療の処方せん

病名	#1 レビュー不全症候群：レビュー軽視症	
発症	開発者 管理者	病原因 開発者：手順無視 管理者：組織文化不具合
症状	<ul style="list-style-type: none"> 変更3点セットを書いているのにレビューを実施しようとしない。また実施しても誰もレビューに参加しない。 時間がないことを理由にレビューを実施しない。 	<p>発症原因</p> <p><開発者></p> <ul style="list-style-type: none"> 変更3点セットの目的は、視点の異なったレビュー機会を設けて当人の気付かぬチーム力で発見することであることではない。 <p><管理者></p> <ul style="list-style-type: none"> もともと一人プロジェクトのスタイルに定着していて、他人の成果物に対する文化がない。 <p><開発者・管理者共通></p> <p>XDDPの変更3点セットは、レビューで合理的に問題を発見することであり、レビューを行わなければ変更3点セットを記述すること自体を理解したうえで、レビューを必ず実施する。（レビューはXDDPの生命線）</p>



AFFORDD公式HP「資料ライブラリ」から入手！



派生開発推進協議会

AFFORDD