【Embedded Technology 2016　スペシャルセッション】

派生開発問題解決セミナー2016

派生開発でアジャイルに対応するために何が必要か

XDDPを活用してアジャイル派生開発に挑む

～変更情報を適切に活用することで混乱を回避できる～

2016年 11月 18日

派生開発推進協議会
代表　清 水　吉 男
URL=http://affordd.jp

株式会社 システムクリエイツ　代表取締役
（新URL）URL=http://soft-koha-hp.la.coocan.jp
（ブログ）http://kohablog.cocolog-nifty.com/
shimz@nifty.com

自己紹介：清水 吉男

- ① 株式会社システムクリエイツ　代表取締役　1983年 設立
  （最初からフリーランス、会社も一人会社）
- ② 派生開発推進協議会(AFFORDD)　代表　2010年2月 設立
- 経歴
  - 1968～1971 オフコン／汎用機の時代（約半数は失敗）→ 撤退（１年後に復帰）
  - 1972～1977 オフコン／汎用機の時代（業務系、オンライン系）【1973年USDM を考案】
  - 1977～1995 組み込みシステムの時代（ECR、ICE、プリンター etc）【1978年XDDP を考案】
  - 1987～1995 システム設計コンサルティングの時代【1990年頃PFD を考案】
  - 1995～ 今日 プロセス改善のコンサルティングの時代【CMM との出会いが契機】

1973～95年の22年間、仕様トラブルも納期遅れもなし ＋ 営業せず

- 著書（単行本）
  - 【改訂第２版】要求を仕様化する技術・表現する技術（技術評論社）
  - 『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）
  - 「SEの仕事を楽しくしよう」（SRC）
  - 「わがSE人生に一片の悔いなし」（技術評論社：新書）

「派生開発」の必要性

- 「保守」
  - 訂正：バグの訂正
  - 改良：小さな変更
    - 保守マニュアル：想定される変更の対応方法を整備

社内の業務システム自体は、
他社と競争していない

- 組み込みシステムで起きていること
  - 製品は競争に晒されている
  - 競争に勝つための機能追加と変更
  - 事前に変更箇所などを想定できない

従来の「保守」
プロセスでは
対応しきれない

派生開発プロセス（XDDP）を公開
同時に「USDM」と「PFD」も公開

派生開発推進協議会：AFFORDD

- **2010年２月に「派生開発推進協議会」（http://affordd.jp） を設立し、「XDDP」「USDM」「PFD」などの派生開発に有効な方法の研究と普及**

カンファレンス

次回＝2016年5月27日 (金)

- 年１回開催、一般公募
- 現場での成果／研究会の成果を発表
- 2016年：大阪での開催を予定

研究会／フォーラム

- 現在12テーマの研究会が活動
- XDDPやUSDMに対して、入門から効果的な活用方法について研究活動
- 研究会の成果をカンファレンス等で発表
- 次の指導者の育成

勉強会

- XDDP，USDM, PFDを多くの人に知ってもらうために、首都圏以外に地方でもセミナーや勉強会を繰り返し開催する

地方部会

- 各地方単位で会員がまとまって情報交換や技術交流をすすめる

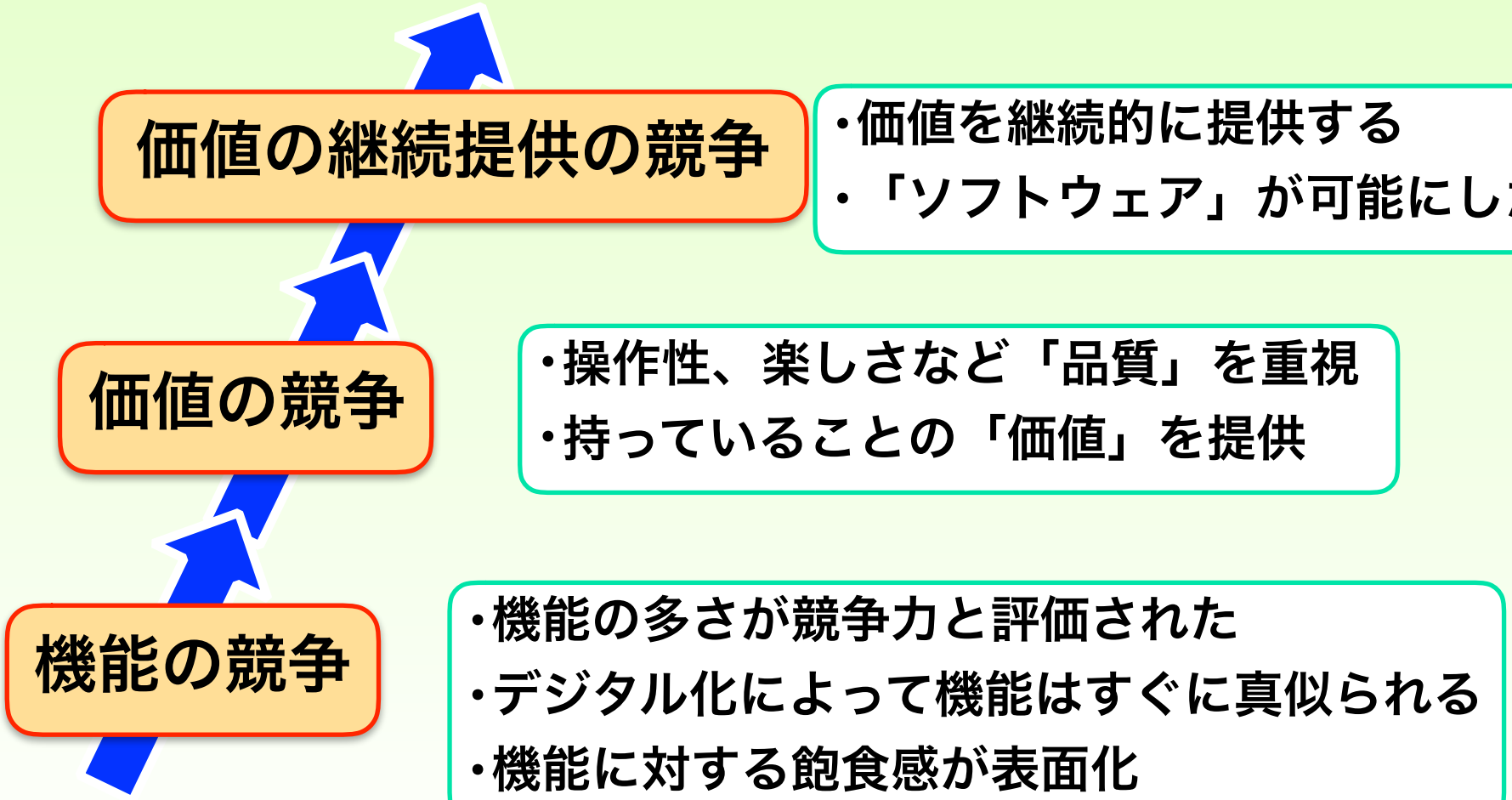
地方支援

- 地方を強くするという考えのもとに、地方行政の活動を支援する

IoTの時代に・・・
派生開発もアジャイルにできないか

ソフト開発における市場の要求の変化

・QCDの要求と合わせて競争のキーワードが変化している



その延長上で「IoT」のうねり

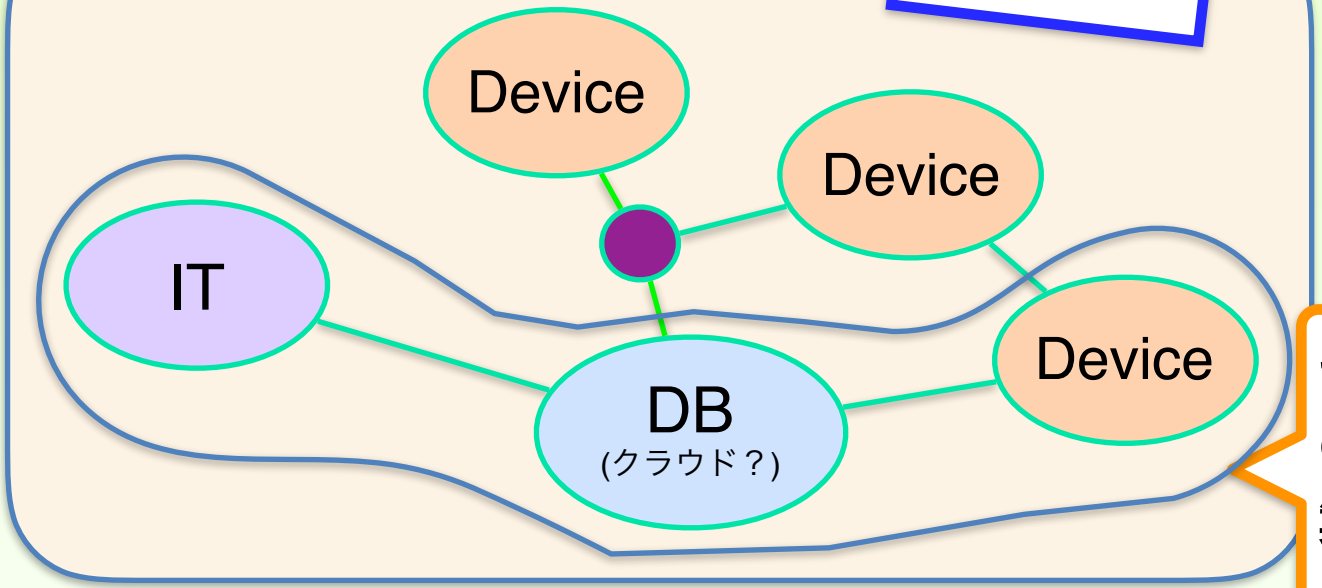
- ETの会場も「IoT」の花盛り！

IoT

IoT

IoT

簡単な「IoT」の構成イメージ



それぞれの領域（複数の領域）でいろいろな製品やシステムを作る

問題は、それらの製品やシステムを「どう作るか」である

市場の変化は待った無し！

我々の都合なんか聞いてくれない！

あらゆる製品が「Software Defined」に

✦製品を「Software Defined」に作ることで・・・

・メーカー

早く買ってもらって後悔させない

・顧客

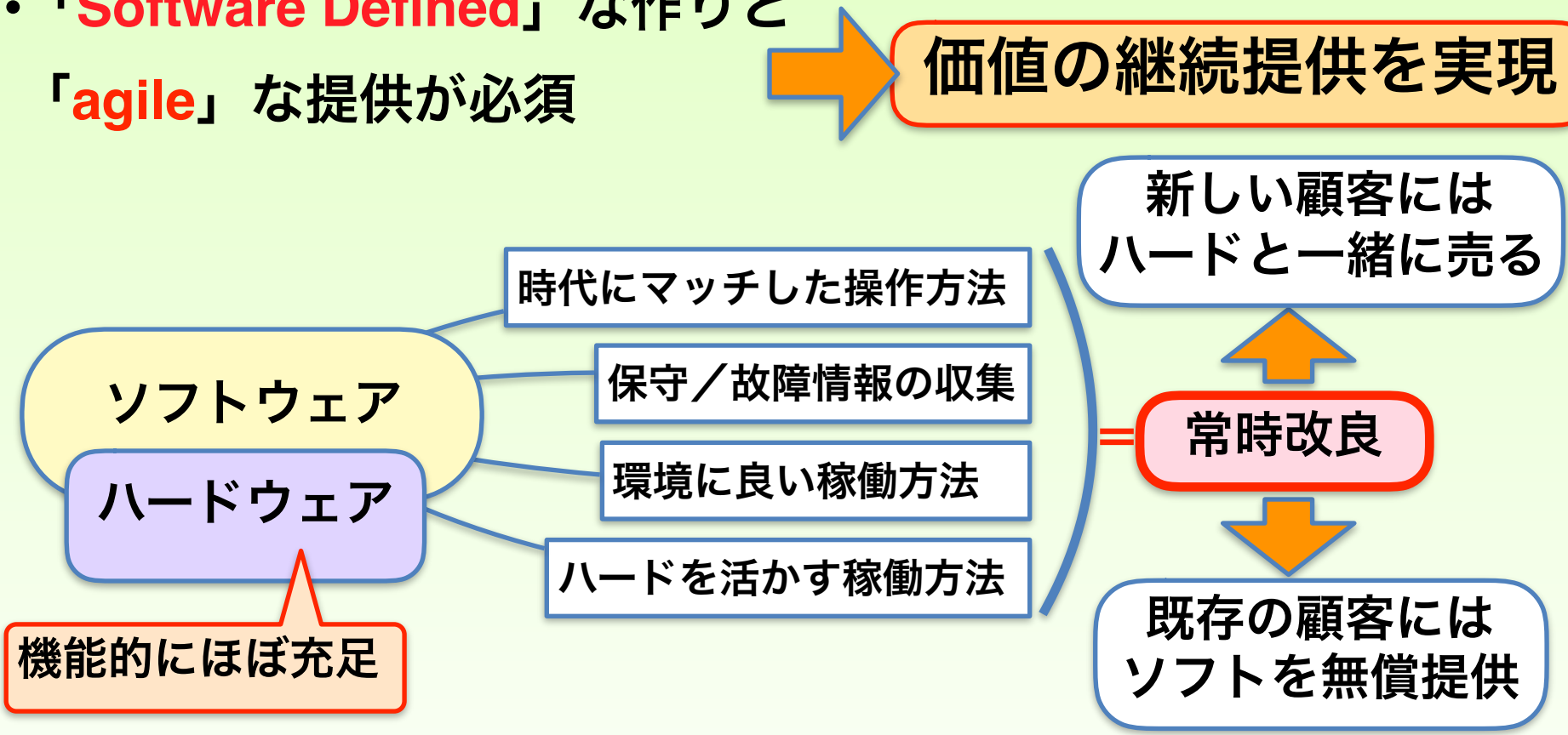
早く買って後悔しない

- **継続的**な価値の提供
- **更新**したソフトを無償提供
- **プラットフォーム**の提供
- 価格競争に巻き込まれない

HP社など：IT系のソフト部門は売却したが、「Software Defined」の開発部門は残した

「Software Defined」と「agile」で競争に勝つ

- 「Software Defined」な作りと「agile」な提供が必須



ここに「信頼」が加わると強烈な競争力になる

同じことが派生開発にも求められている

- 新規開発を「Software Defined」や「agile」に作る以上、
  派生開発にも・
  - **「Software Defined」を維持しながら**
  - **「agile なリリース」の達成**

  が求められる

これによって

デジタル化の時代にあって
「ソフトウェア」で顧客を囲い込み続けることができる！

どうやって
実現するか？

ただし、ソースコードの状態によっては
簡単にはいかない

ところが・・一つ厄介な問題が残っています

- 今でも「ソフトウェア」に理解のない トップが少なくない

「IoT」＝「モノ」がネットに繋がっただけという認識

Internet of Things

「派生開発」は所詮 “ちょい変” だろ！

だったら、コストの安いオフショアで開発だ！

Software Defined

agile

その結果・・・

ソースコードは著しく劣化

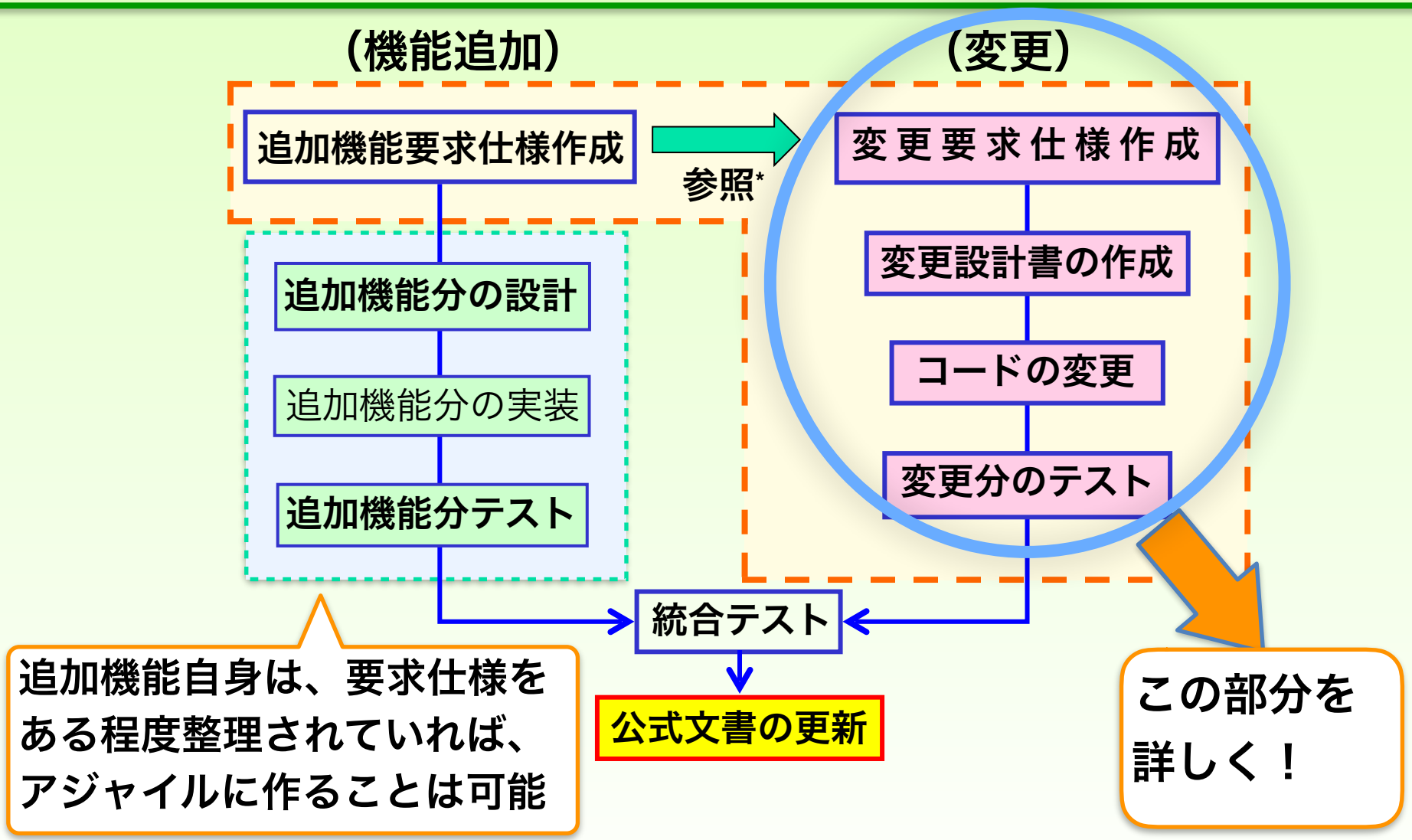
組織の開発能力の空洞化

XDDPで派生開発を
アジャイルに
開発することは可能か？

「Software Define」は別の機会に・・・

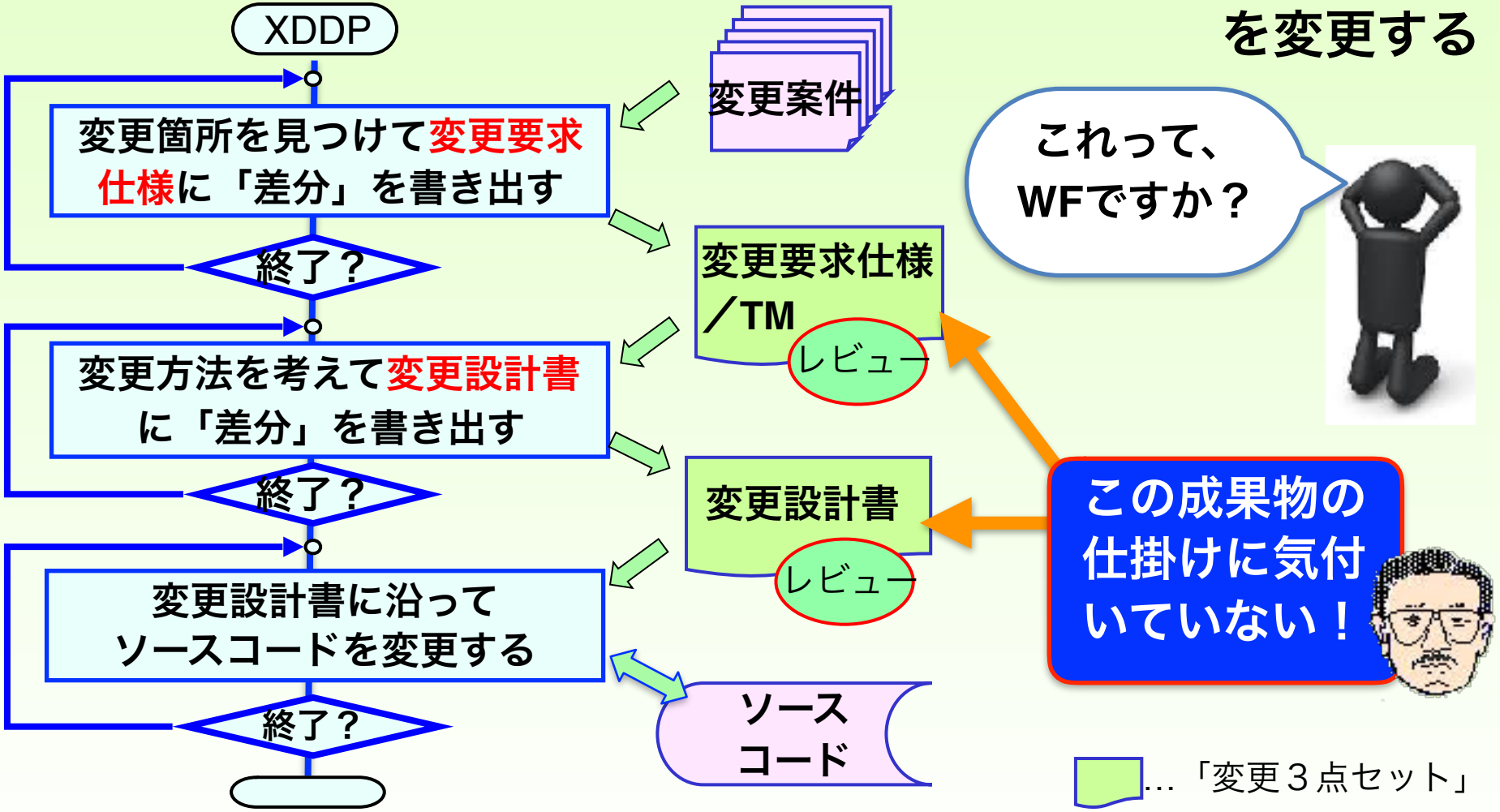
本日は、派生開発を「Agile」に開発することついて考えましょう！

XDDPは機能追加と変更のプロセスを分ける



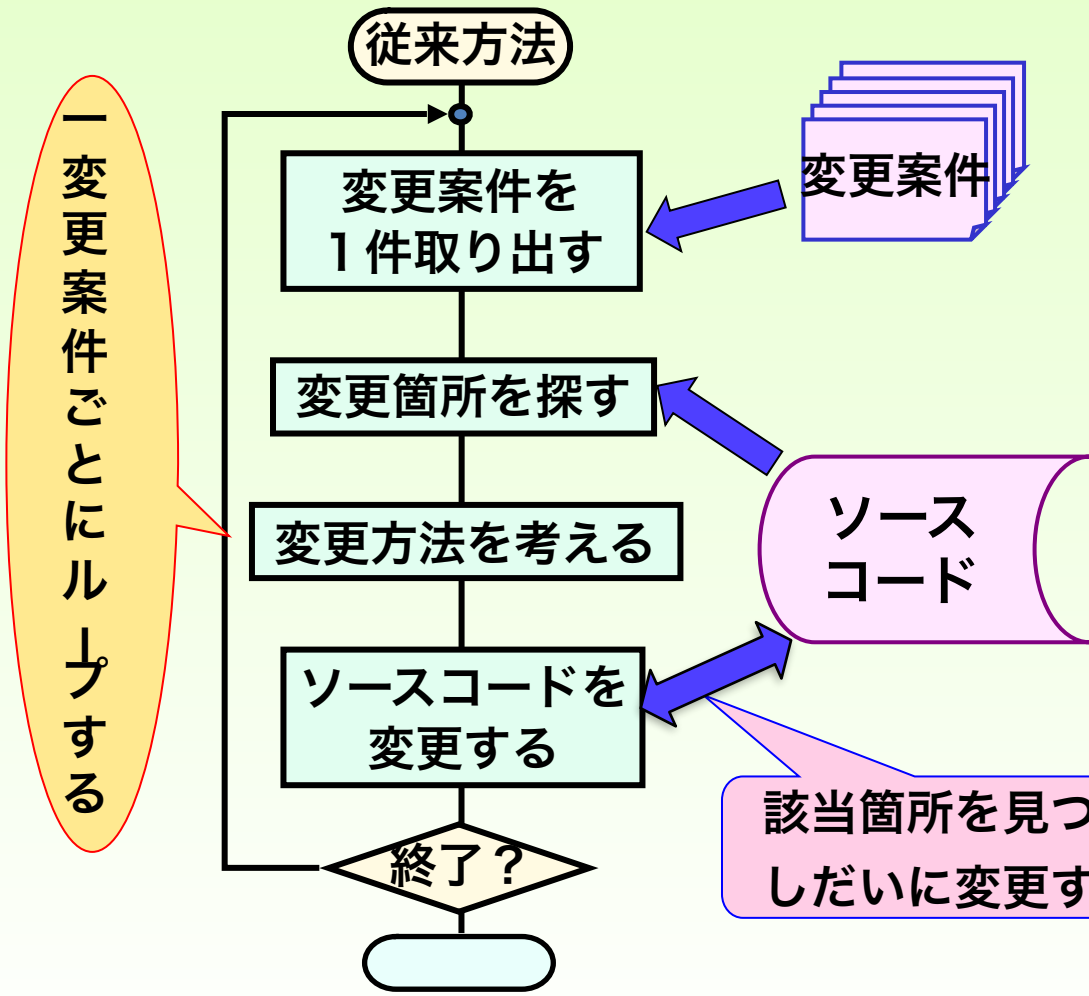
XDDP（変更）はウォーターフォールか？

- XDDPの変更では３種類の成果物を生成しながらソースコードを変更する



一般に行われているのは何？

- １つの変更案件ごとにソースコードを変更する



じゃ、
これは何と呼ぶの？

無駄な成果物は作っていない！

素早くソースコードを変更！

“アジャイル”って
いうつもり？

XDDPで
は、どうしてソースコードを
最後にまとめて変更するのか
わかりますか？

ソースコードの状態が悪い

影響箇所が把握できない

あとからの変更で、すでに
終わったはずの箇所を覆す

だから

すべての変更箇所を把握した上で、ソースコードの変更に取り掛かるしかない

ただし、
この状態では、派生開発をアジャイルにできない！

派生開発の現実

- これまでの無規律な変更でソースコードが傷んでいる

変更要求仕様／TM

#	変更要求・仕様		A	B	C	D	E	F	G	H
5	画面に通信記録の表示を追加する									
	5.1	接続状況の表示の大きさを・・に変更する				F1				
	.	・・・								
	5.4	表示用メモリの配置を・・に変える				F1				
	5.5	受信時のエラー処理を変更する		F9			F3	F4		

XDDPで取り組むことで様子が残る

- 本来の変更仕様
- その変更の影響による変更仕様

　　影響のパターン　→　拡散の懸念

などが見える

- 変更先の拡散　←　コピペの結果？
- 同一関数での異なる変更　←　凝集度？

などの様子が見える

この状態では、ソースコードの変更を遅らせるしかない

XDDPで
どうすれば、アジャイル
に対応できるのか

ヒント１

「XDDP」では、ソースコードを
最後に変更することは必須ではない

ソースコード自身の
事情による

ソースコードに “秩序” があれば、
最後まで変更を遅らせる必要はない

XDDPの「変更３点セット」がこれを可能にする

ヒント２

これを活用する！

「変更３点セット」は、
ソースコードの状態を写している

ソース
コード

XDDPの「変更３点セット」
の情報

リファクタリングなどで
“秩序” を回復させる

- 変更の性質
- 変更の影響の様子
- 変更箇所の拡散状況

などがわかる

＋ バグ情報

変更範囲を判断
するための情報

前置きはここまで

つづいて
パネルディスカッション
をお楽しみください

XDDPを活用して派生開発を
アジャイルに取り組む方法