派生開発とシステムテスト

派生開発推進協議会
T4　研究会

永田　敦

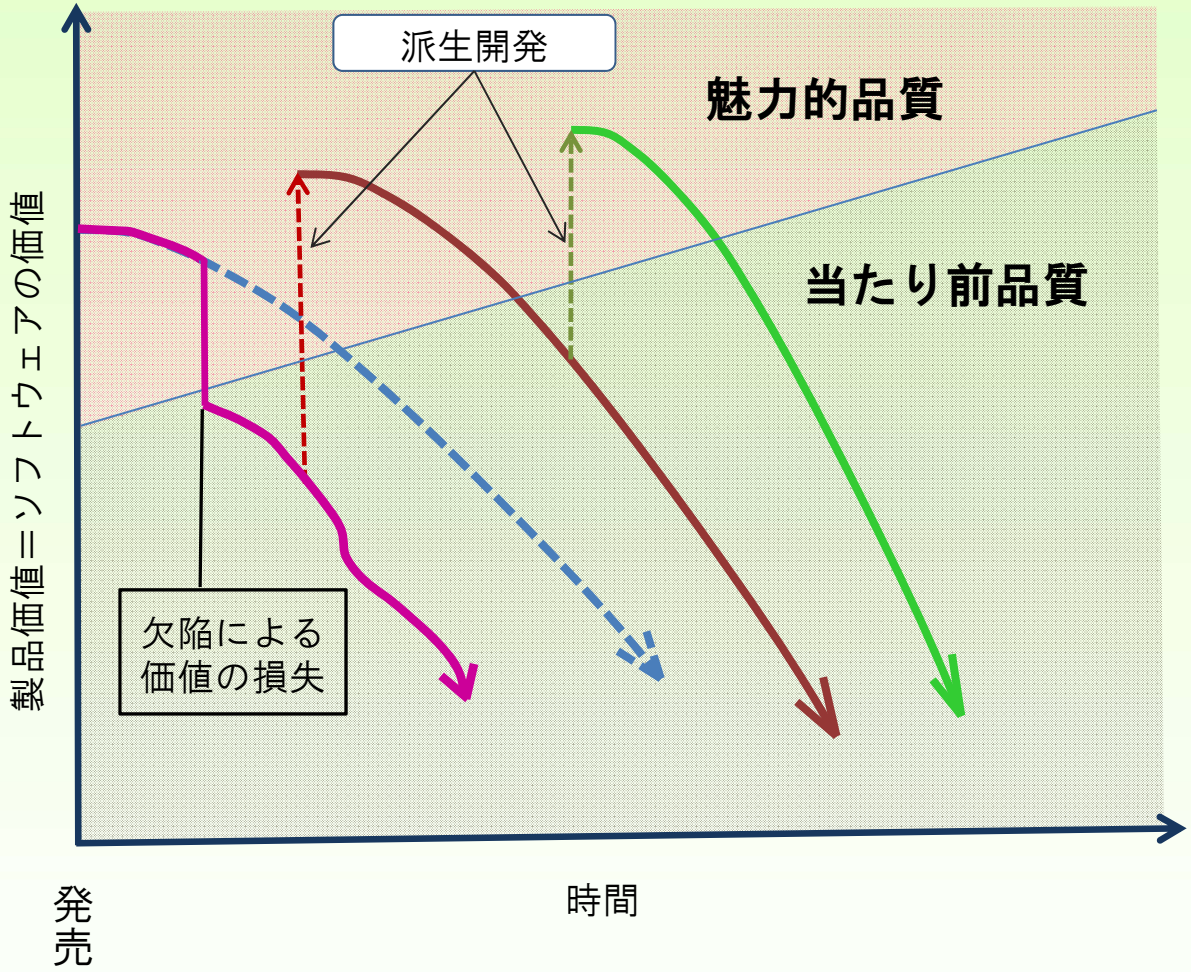
研究会では、派生開発でソフトウェアテストを成功させるアプローチや手法の確立を目的として活動している。

組織によっては、開発の90%以上が派生開発

派生開発

- 目的
  - 新しいモデル（製品）に使用し早く出荷する
  - 新機能の追加、アップグレード
- 前に動いていたコードを使う
  - 動いているコードを変更する
  - 動いているコードに新規のコードを追加する
  - 動いているコードに他から動いているコードを移植する
- 保守としての変更も含まれる
  - バグフィックス

派生開発における製品価値



派生開発の問題(開発編)

- 変更自体は小さい
  - 数行
  - 簡単に早く終わるものと思ってしまう
- 前に動いていたコードを使う
  - 動いているコードを変更する
  - 動いているコードに新規のコードを追加する
  - 動いているコードに他から動いているコードを移植する
- 保守としての変更も含まれる
  - バグフィックス

「変更3点セット」の成果物の意味

- 「XDDP」の変更プロセスでは3つの成果物を作成する（必須）

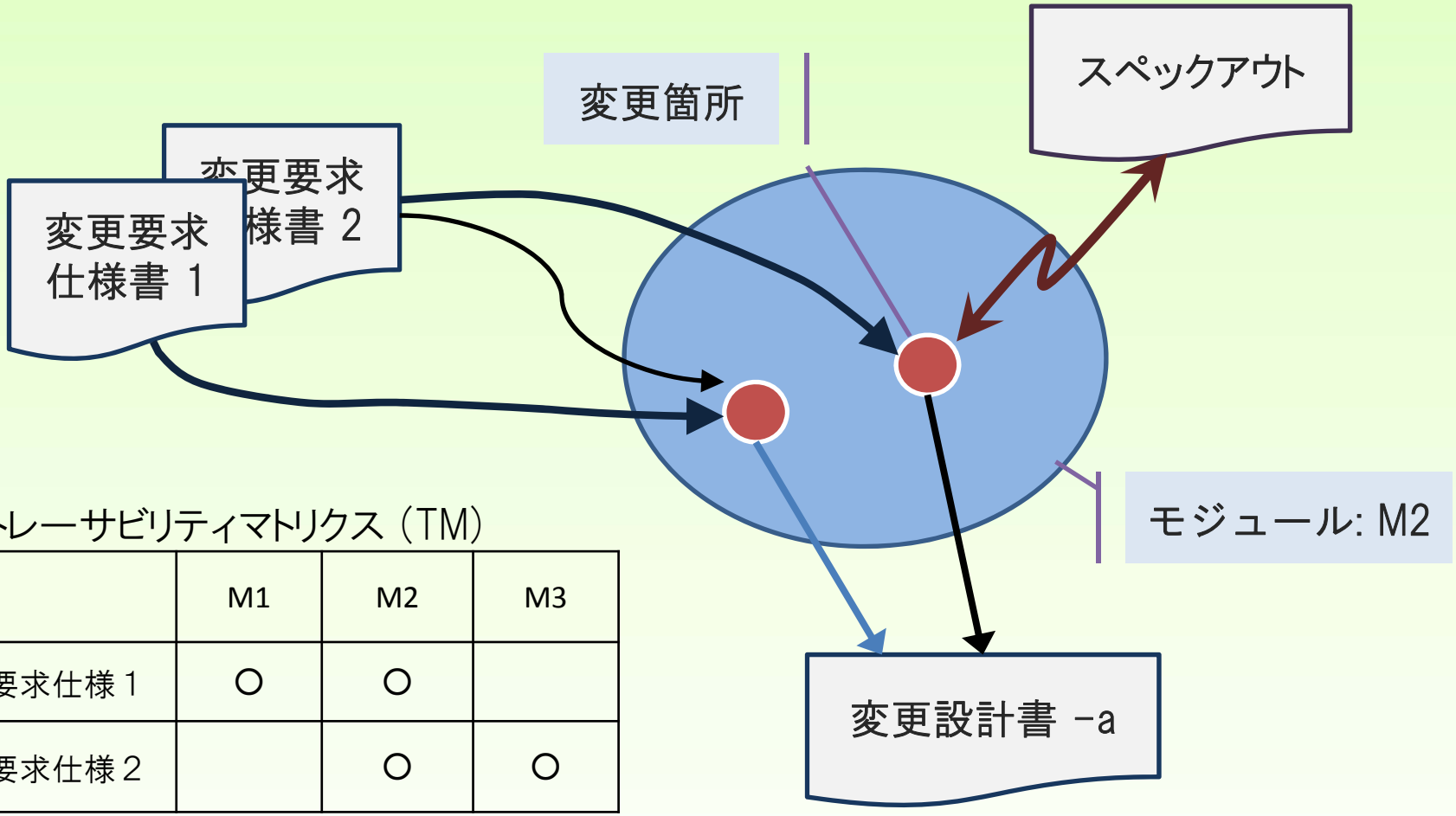
成果物	カバー範囲	記述内容	レビュー機会	
変更要求仕様書	What （Why）	何を変更するか？ どのような振る舞いを変更するか なぜ、変更するのか？	○	○
TM（Traceability Matrix）	Where	変更する仕様がどこにあるか？	○	
変更設計書	How	具体的な変更方法を記述する	○	○

- 効果的なレビューの機会を確保して、「部分理解」の中で生じる担当者の思い込みや勘違いをカバーする
- ソースコードの変更作業と、公式文書のマージの両方に活用できる
- 今回の派生開発の変更記録として保存する

変更要求仕様書：USDM

要求	MAL01-03	検索結果を扱いやすく表示し、そこから選択したい
	理由	目的のメールが一つとは限らないので、絞り込めるような操作がしたい
□	＜検索結果の表示＞	
□	MAL01-03-1	検索されたメールの件巣を一覧の上に表示する
□	MAL01-03-2	該当するメールが存在しないときは「該当無し」を表示する
	＜検索メールの表示＞	
□	MAL01-03-5	検索されたメールの「Subject」を一覧で見せる
□	MAL01-03-6	検索されたメールに連続番号をつけて表示する
□	MAL01-03-7	検索されたメールの件数１０件を超えるときはスクロールバーを見せる
	＜メールの中身の表示＞	
□	MAL01-03-10	一覧から１つのメールを選んで、その内容を見ることができる
□	MAL01-03-11	開示されたメールの中にある検索キーワードと一致している文字列を赤色で表示する
	＜メールの選別＞	
□	MAL01-03-15	不要なメールを選んで一覧から消すことができる
□	MAL01-03-16	一度不要として消されたメールを復活することができる

変更要求仕様書／ＴＭ／変更設計書

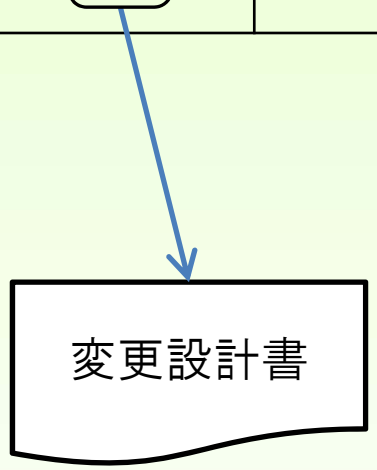


トレーサビリティマトリクス（TM）

	M1	M2	M3
要求仕様１	○	○	
要求仕様２		○	○

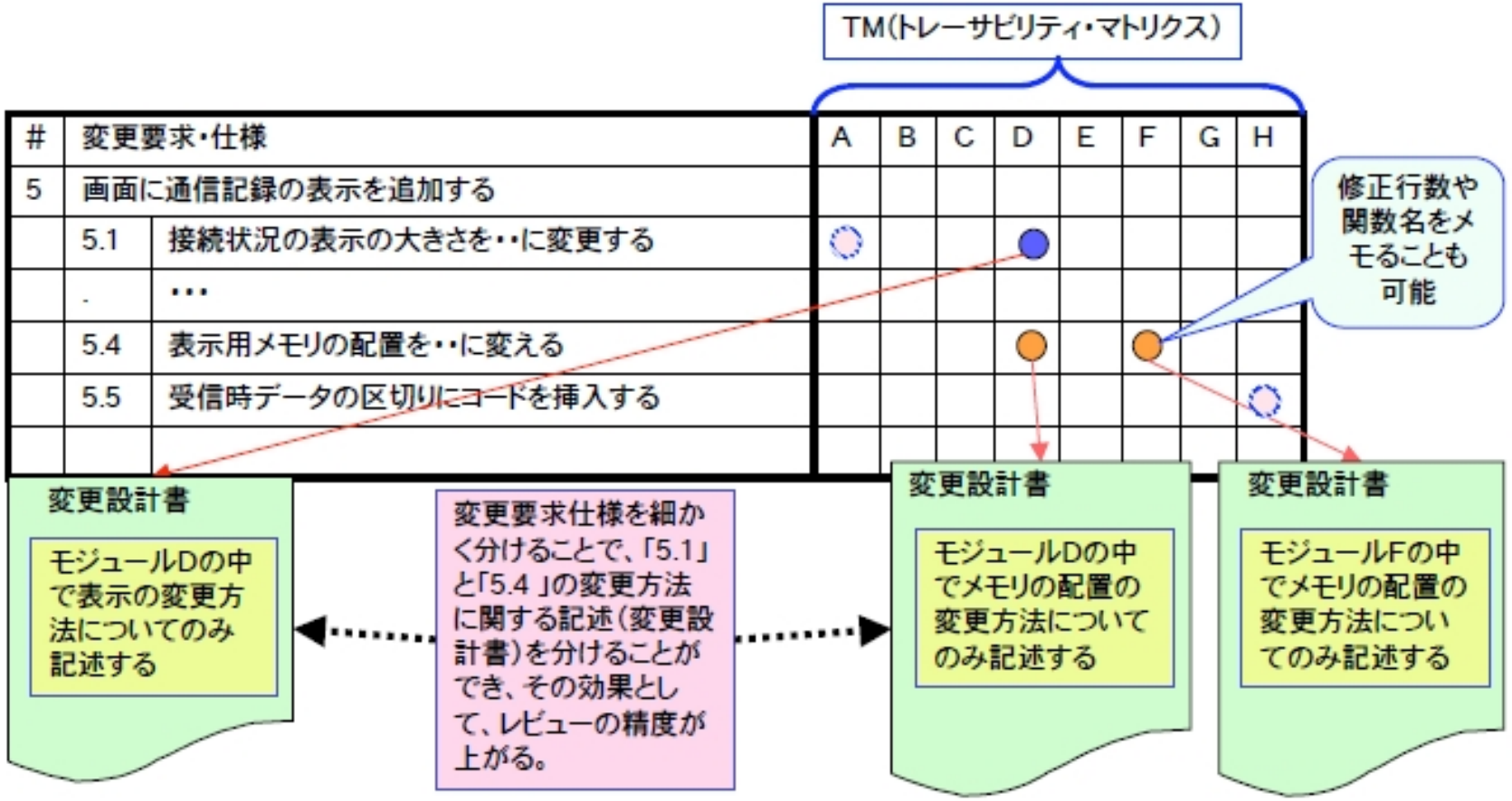
要求仕様書，ＴＭ，変更設計書 その２

	module 1	Module 2	Module 3	Module 4	Module 5
変更要求仕様１	○		○	○	
変更要求仕様２			○		○

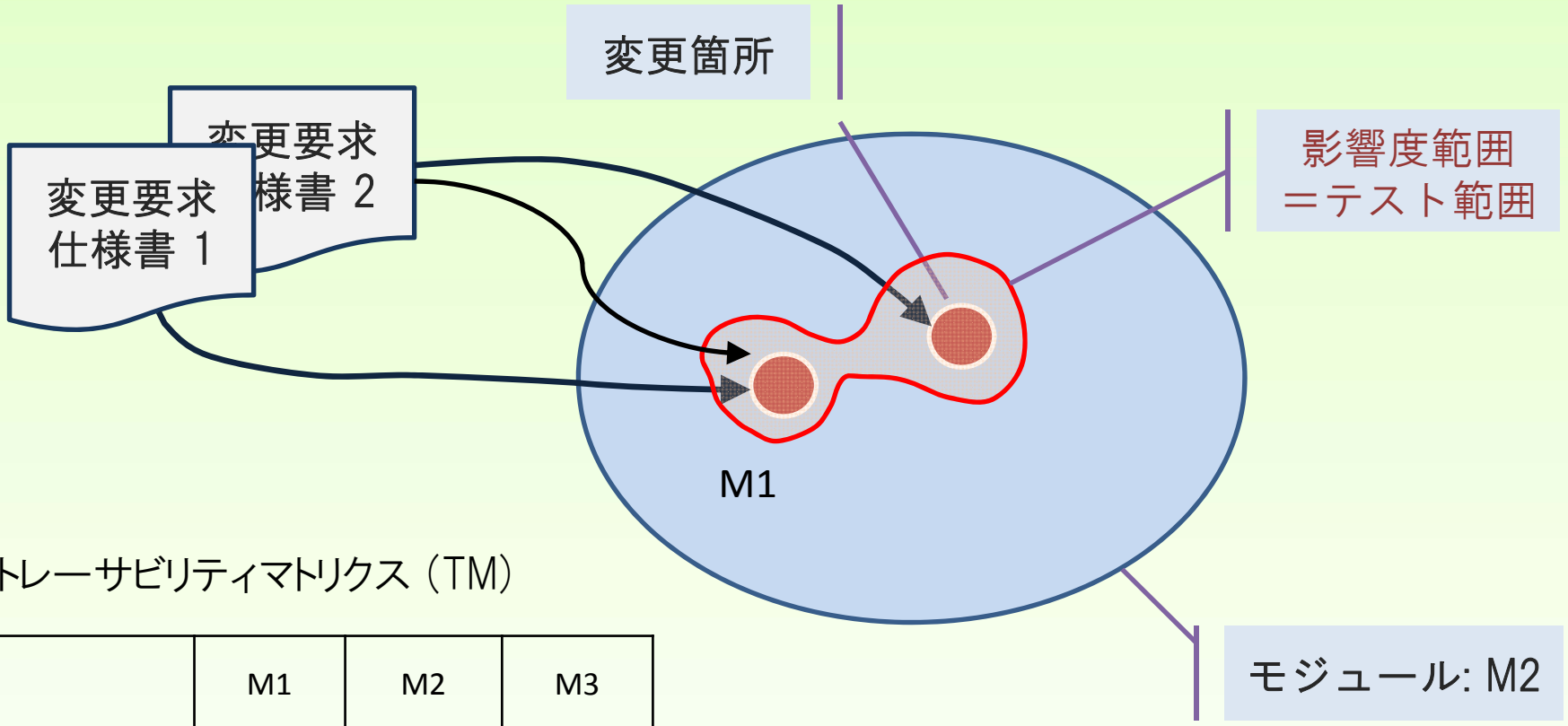


TMを介して変更設計書と繋ぐ

- TMは「変更要求仕様」と「変更設計書」の蝶番の役目を果たす



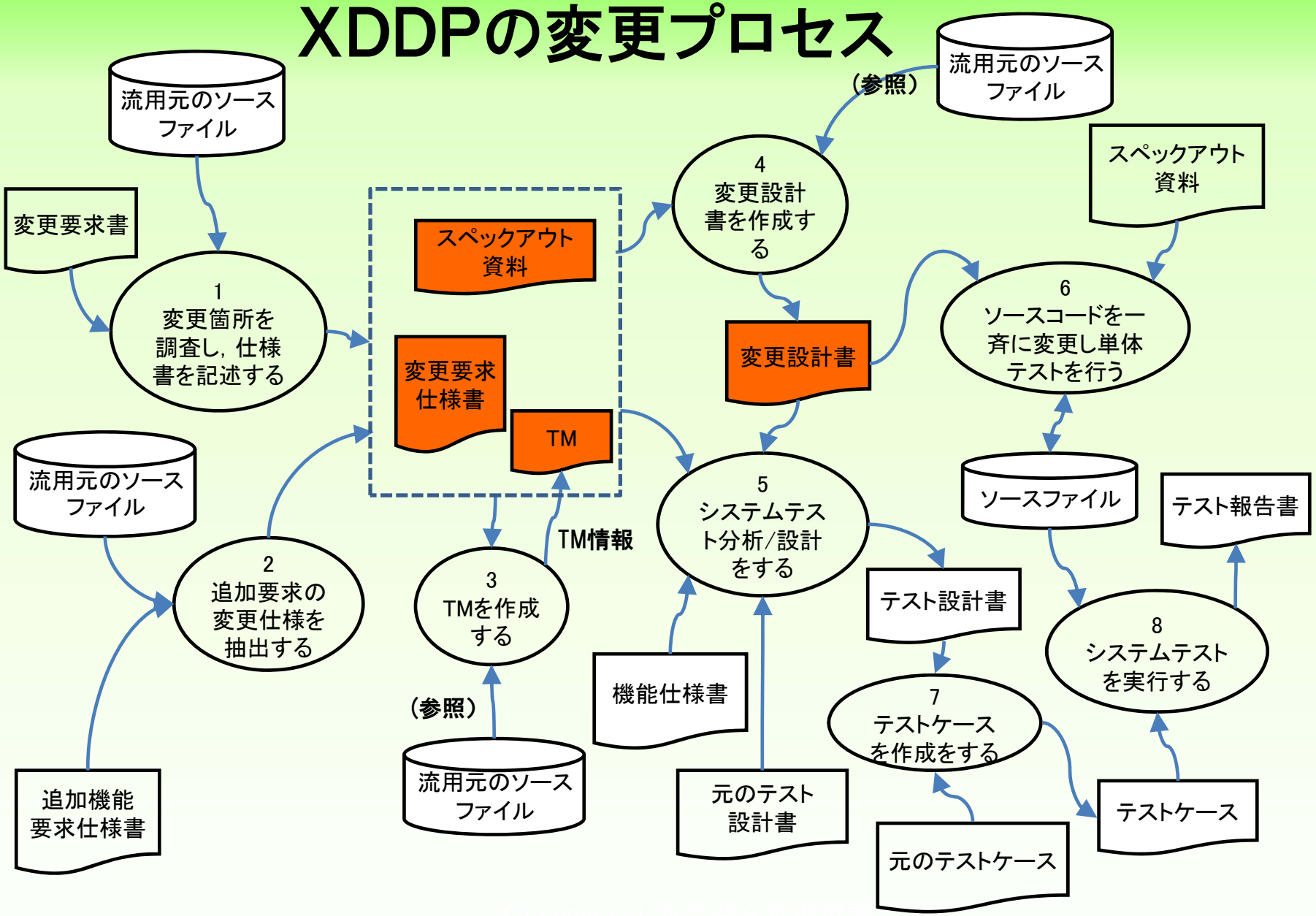
変更の影響度



トレーサビリティマトリクス（TM）

	M1	M2	M3
要求仕様１	○	○	
要求仕様２		○	○

XDDPの変更プロセス



変更プロセスの簡単な説明

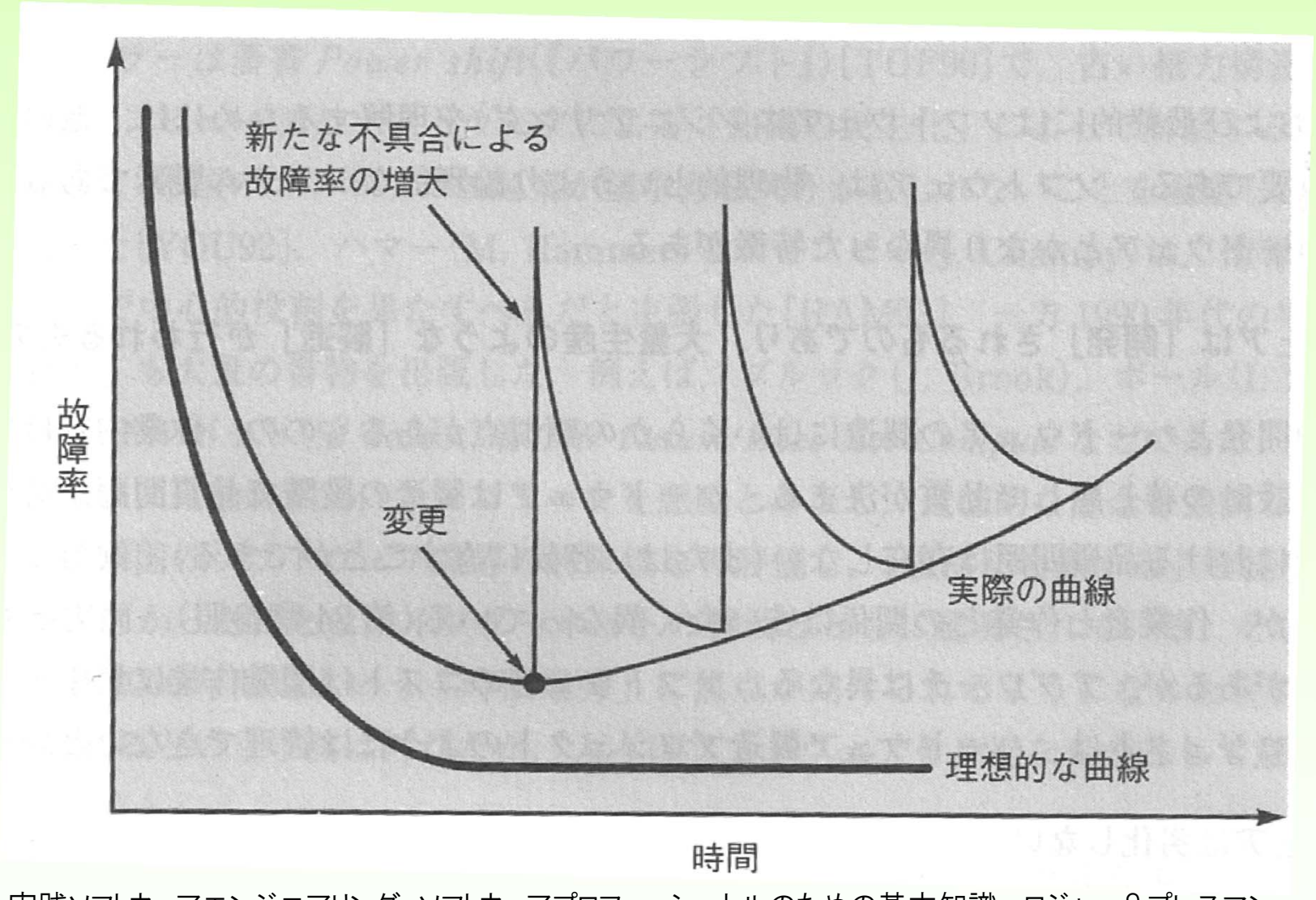
この単位で作業が分離される

- 1.1 変更箇所を仕様書に記述する
  - 機能仕様書などの文書から変更箇所(仕様)が特定され、変更仕様書の該当する箇所に記述する
- 1.2 変更箇所についてソース等を調査する
  - それぞれの変更要求に対してソースコードを調査して変更箇所を探し、発見した箇所を変更仕様として変更仕様書の該当箇所に記述する。その際の調査資料をスペックアウト資料として残す。
- 1.3 追加要求の変更仕様を抽出する
  - 追加機能を受け入れるための変更箇所(仕様)を探し、それを変更仕様書の該当箇所に記述する。ソースコードの調査結果は資料として残す
- 1.4 変更要求TMを作成する
  - すべての変更仕様に対して具体的にソースコードとの対応付けを行う
- 1.5 変更設計書を作成する
  - 変更仕様のレビューを終えた後、改めて具体的な変更方法(差分)を記述する
- 1.6 ソースコードを一斉に変更する
  - 変更設計書のレビューを終えた後、ソースコードを一斉に変更する

派生開発の課題

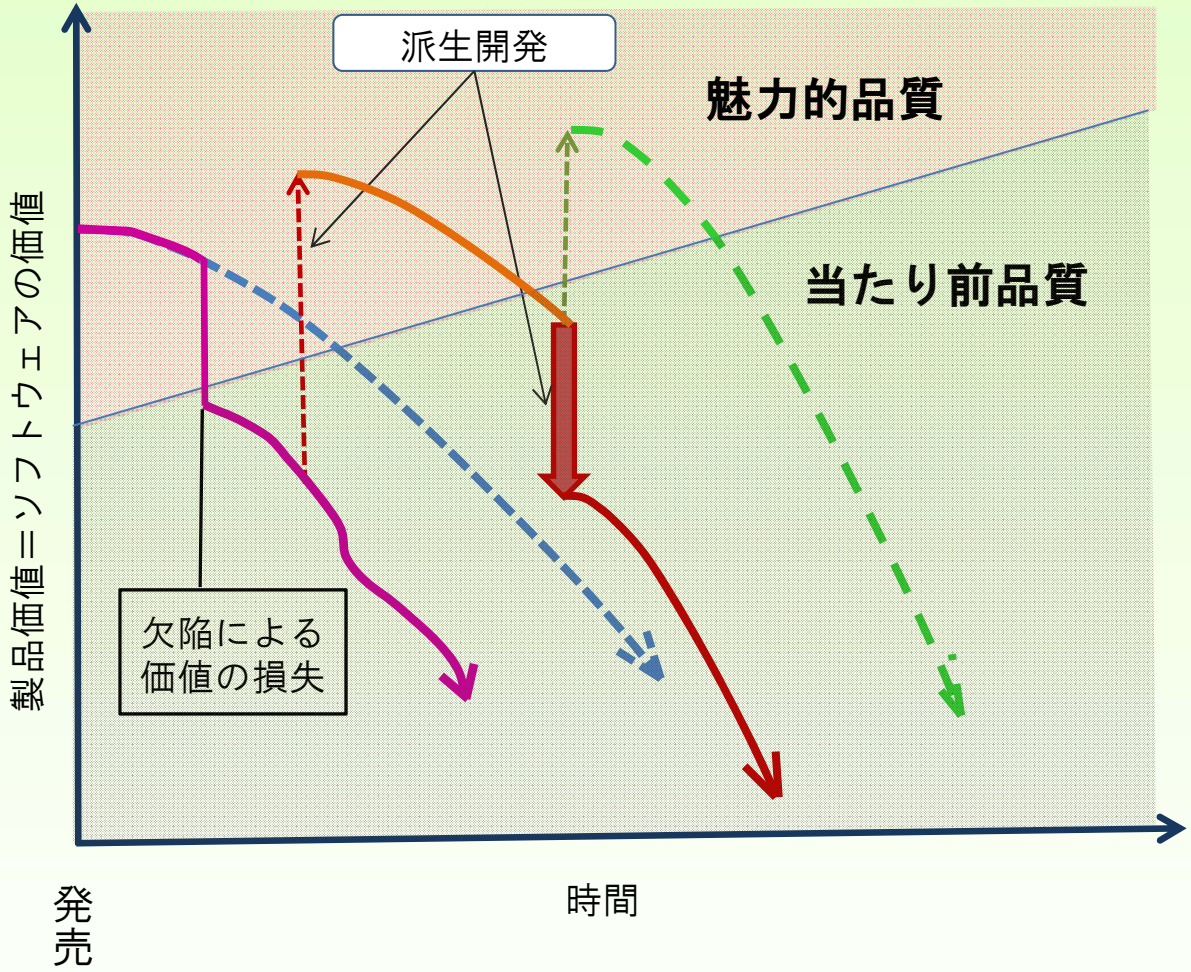
- 変更自体は小さい
  - 数行
  - 簡単に早く終わるものと思ってしまう
- 前に動いていたコードを使う
  - 動いているコードを変更する
  - 動いているコードに新規のコードを追加する
  - 動いているコードに他から動いているコードを移植する
- 保守としての変更も含まれる
  - バグフィックス

ソフトウェアの理想的なおよび実際の故障率曲線



実践ソフトウェアエンジニアリング−ソフトウェアプロフェッショナルのための基本知識,、ロジャーS.プレスマン

派生開発における製品価値



課題

- 派生開発におけるテスト
  - 変更されたところ→変更要求仕様
  - 追加されたところ→追加機能要求仕様
    →追加変更の要求仕様

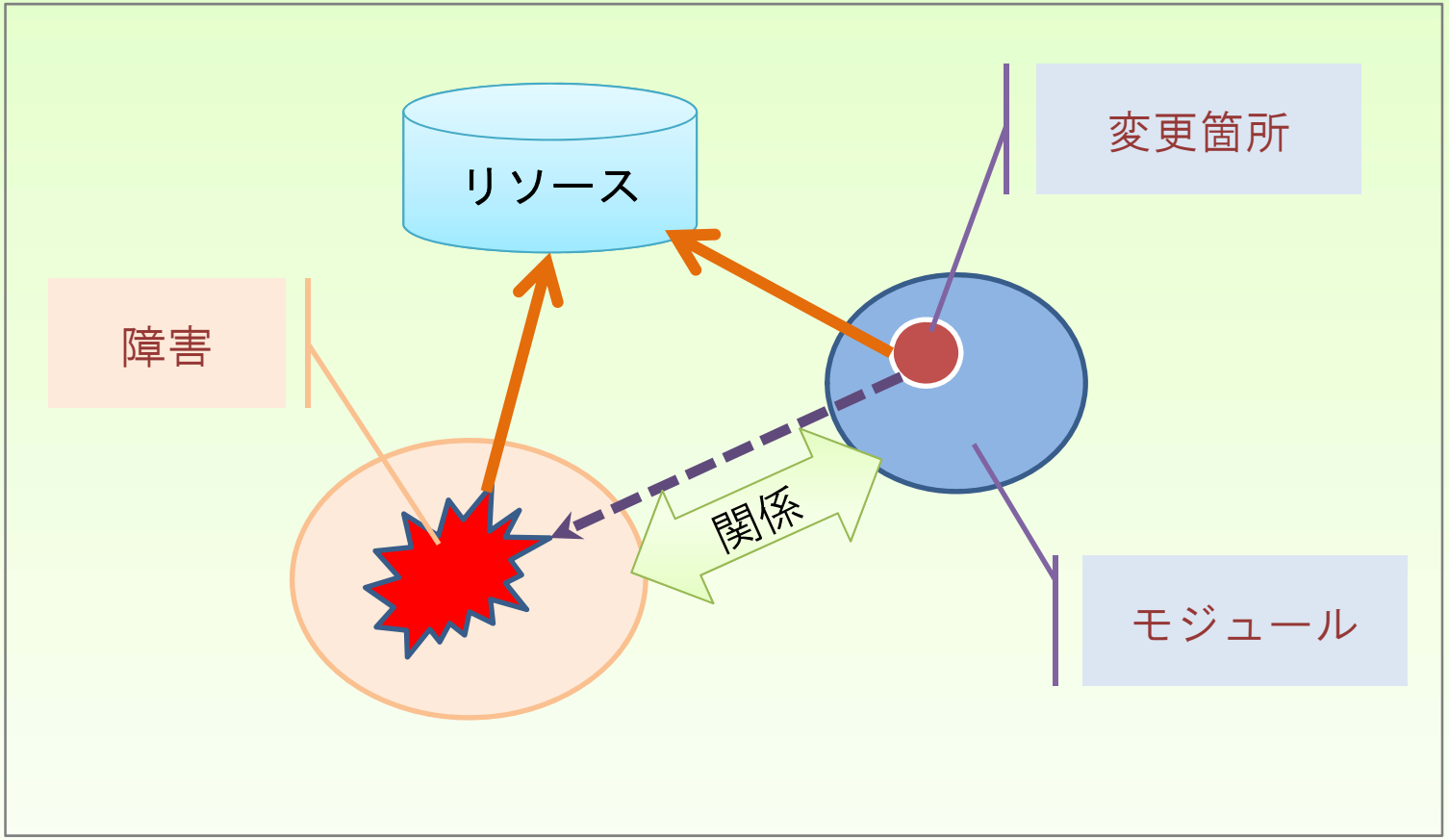
テストベース

テスト設計
変化したところをテストしていく

  - デグレード　　変更による影響範囲

リソースにかかわる問題　競合，衝突

デグレード



システムテストにおける派生開発の問題

- ＸＤＤＰでは、変更するための差分情報に絞って開発を実施する
  - 変更の影響による、振る舞いの変化が見えづらい
  - ＸＤＤＰの成果物だけでは適切なテスト範囲の設定が難しい
- ＸＤＤＰの成果物は最終成果物（ソースコード）を修正するための記述になっている。
  - それらの成果物が、既存のテスト仕様書の変更に用いるには適していない場合がある。

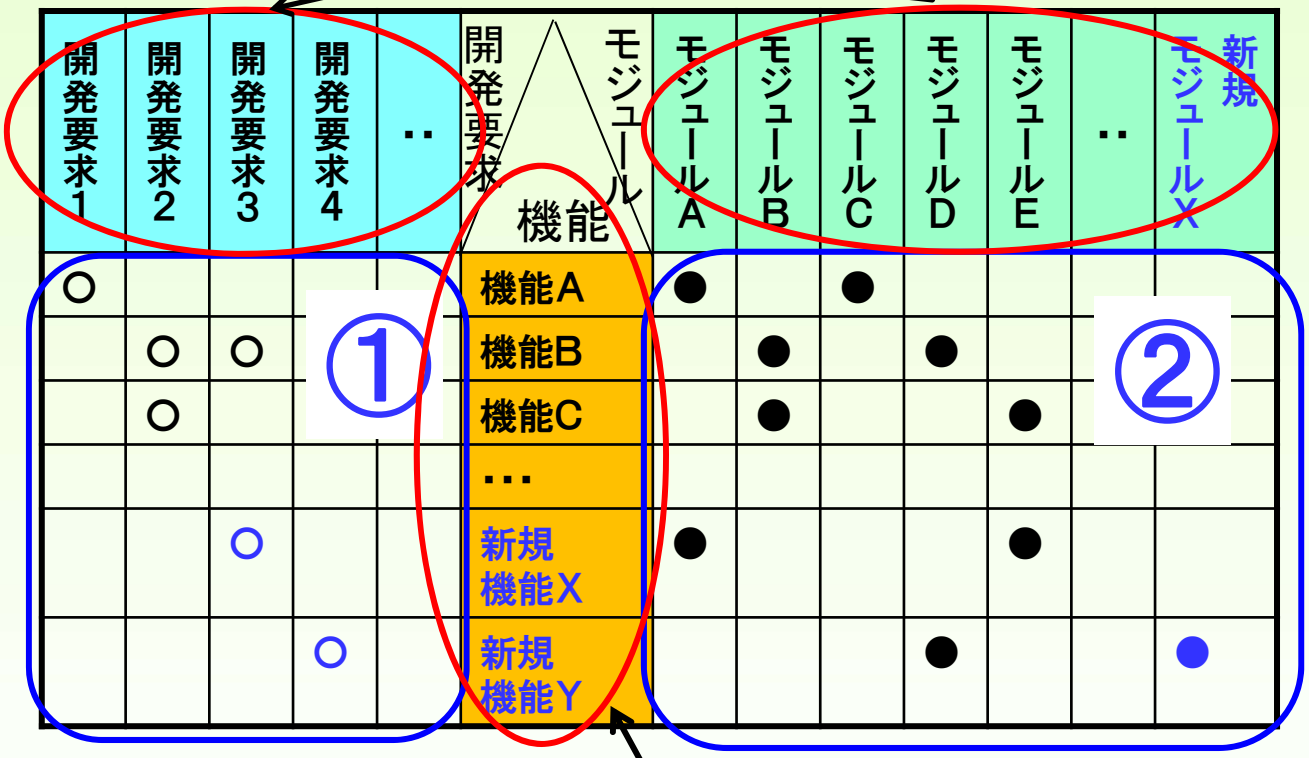
T型マトリクス用いた変更要求の提案　(2011年)

■T型マトリクスを用いた(要求ー機能ー構造)の明確化

「列」に開発要求(左)、ﾓｼﾞｭｰﾙ(右)および「行」に機能項目(＝テスト項目)を配置することにより、機能との関連を明確にし、開発者とテスト技術者の認識を合わせる

今回開発で対応する開発要求
(開発者視点)

変更要求仕様書のTMに同じ
(開発者視点)



既存のﾃｽﾄ項目から抽出した機能項目　(テスト技術者視点)

2つのマトリクスを結合

①開発要件ー機能マトリクス
※製品仕様上の関連を把握

②機能ーモジュールマトリクス
※ソフト構造上の関連を把握

システムに対する影響の明確化

「変更要求仕様書(TM付)」と「T型マトリクス」を連携させ、今回開発要求に対する影響範囲を明確化する

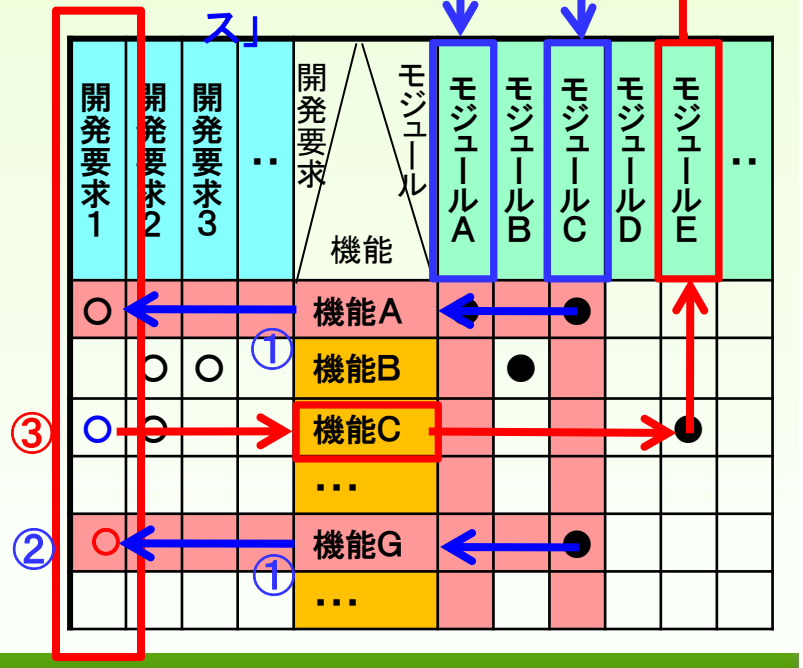
「変更要求仕様書」

TM(トレーサビリティマトリクス)

#	開発要求1 変更要求・仕様		モジュールA	モジュールB	モジュールC	モジュールD	..
xx5	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〇〇に変更する	F1( )				
		...					
	5.3	表示用メモリの配置を〇〇に変える	F2() F3()		F4()		
	5.4	接受信時ﾃﾞｰﾀの区切りにｺｰﾄﾞを挿入する			F5( )		

マトリクスの連携

「変更要求T型マトリクス」



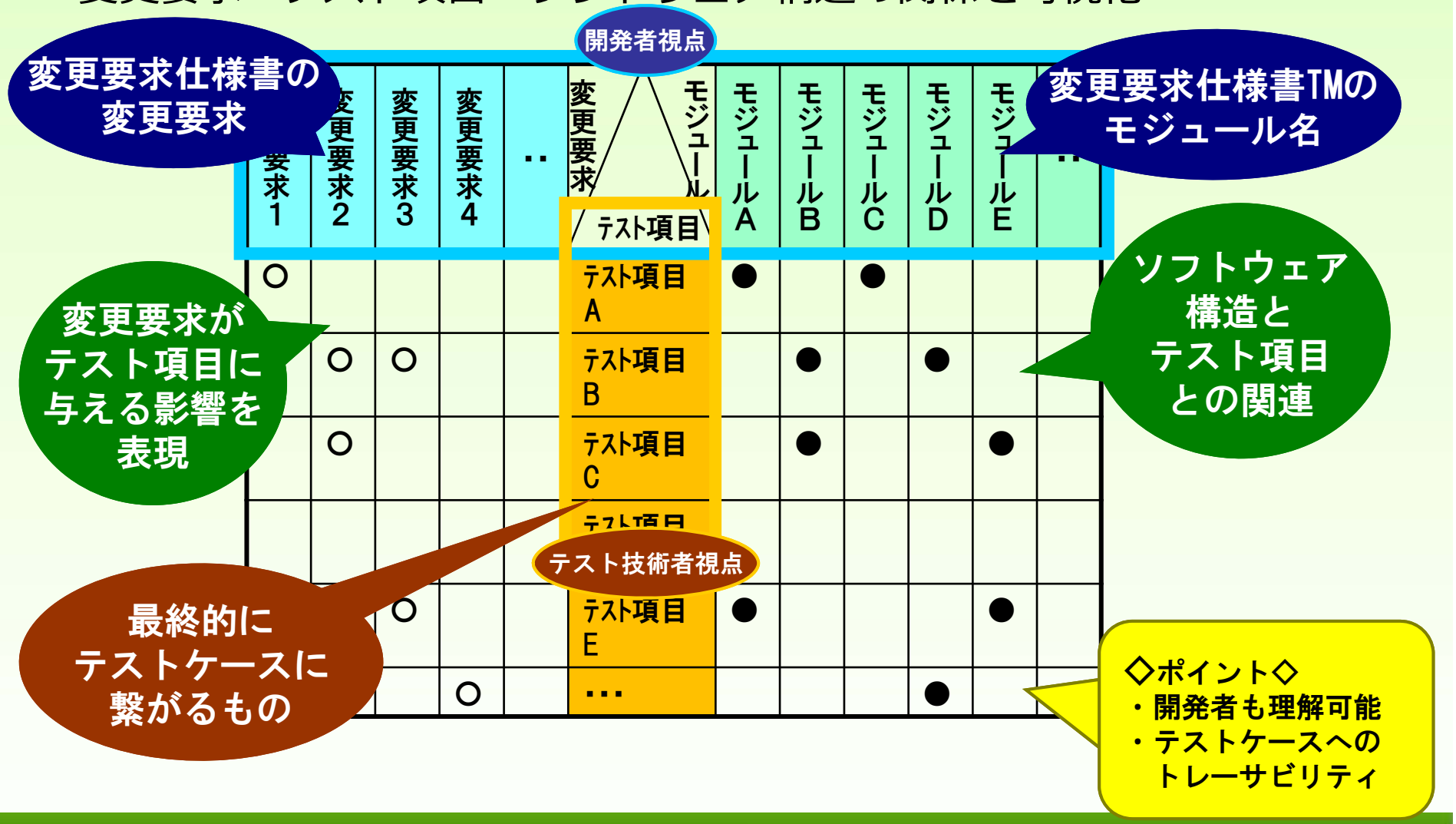
① 変更要求仕様書TMから、開発要求１は、機能Aと機能Gに影響があることがわかる

② 開発要求１と機能Ｇの関連にモレが判明したため、マークを追加する。

③開発要求１と機能Ｃへの関連はあるが、変更要求仕様ＴＭからモジュー ルEへの連携がないため、再度修正モジュール、変更要求仕様を見直す

T型マトリクス (2012年)

- 変更要求ーテスト項目ーソフトウェア構造の関係を可視化

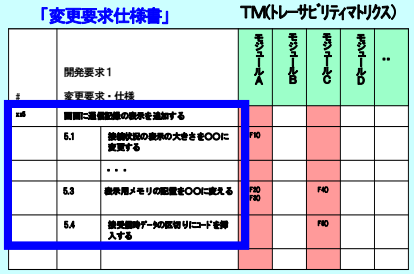


T型マトリクスの構成（1／2）

- 変更要求ーテスト項目の可視化

◆ 変更要求がテスト項目に与える影響を表現
◆ 開発者とテスト技術者の認識合わせに使用

① 変更要求を記載



変更要求仕様書

開発者視点

変更要求1	変更要求2	変更要求3	..	変更要求 ＼ テスト項目
○				テスト項目A
	○	○		テスト項目B
				テスト項目C

テスト技術者視点

② 既存のテスト項目を記載

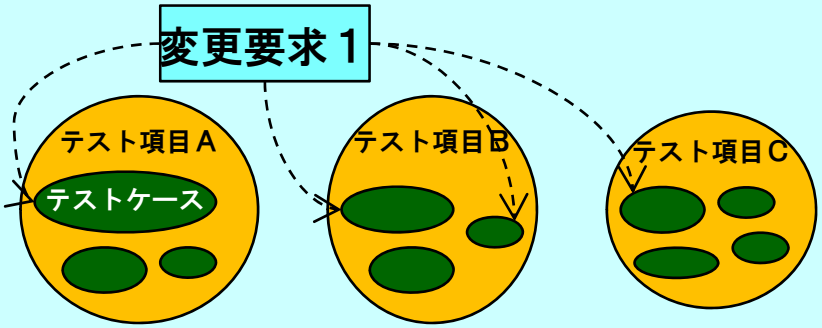
テストケース一覧

1. 沸騰機能	1-1. ・・・
	1-2. ・・・
	・・・
2. 給湯機能	2-1. ・・・
	2-2. ・・・
3. 電源管理	3-1. ・・・
	・・・
4. 異常動作	・・・

他にも・・・
「機能」
「振る舞い」
など

③ 関連する項目に○

※ 変更要求とテスト項目の関係は「1：多」になり、漏れやすい



T型マトリクスの構成（2／2）

- テスト項目ーソフトウェア構造の明確化

テスト項目＼モジュール	モジュールA	モジュールB	モジュールC	モジュールD	モジュールE	..
テスト項目A	●		●			
テスト項目B		●		●		
テスト項目C		●			●	
テスト項目D						

◆ テスト項目がどのモジュールをテストしているか整理
◆ 変更要求仕様書との整合性チェックに使用

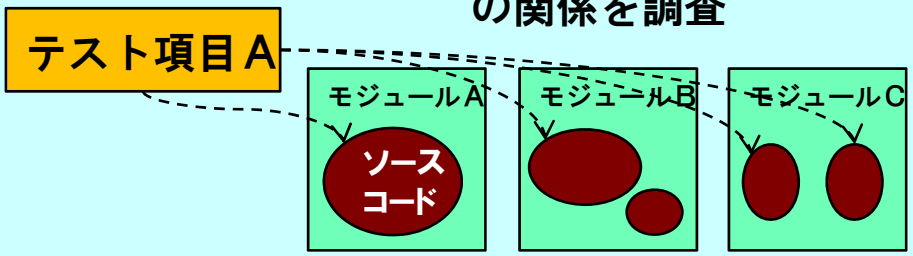
① 変更要求仕様書のトレーサビリティ マトリクス（TM）の列を記載

変更要求仕様書

② 既存のテスト項目を記載

変更要求ーテスト項目のマトリクスと同じ内容を記載

③ 関連する項目に● テスト項目とソースコードの関係を調査



※ 一度作成すると、他のプロジェクトでも流用可

Ｔ型マトリクスの整合性チェック

「変更要求仕様書」と「T型マトリクス」を連携させ、今回開発要求に対する影響範囲を明確化する

２つのドキュメントに矛盾が無いかチェック

「変更要求仕様書」

TM(ﾄﾚｰｻﾋﾞﾘﾃｨﾏﾄﾘｸｽ)

#	開発要求１ 変更要求・仕様		モジュールA	モジュールB	モジュールC	モジュールD	..
xx5	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	F10				
		・・・					
	5.3	表示用メモリの配置を○○に変える	F20 F30		F40		
	5.4	接受信時データの区切りにｺｰﾄﾞを挿入する			F50		

チェック

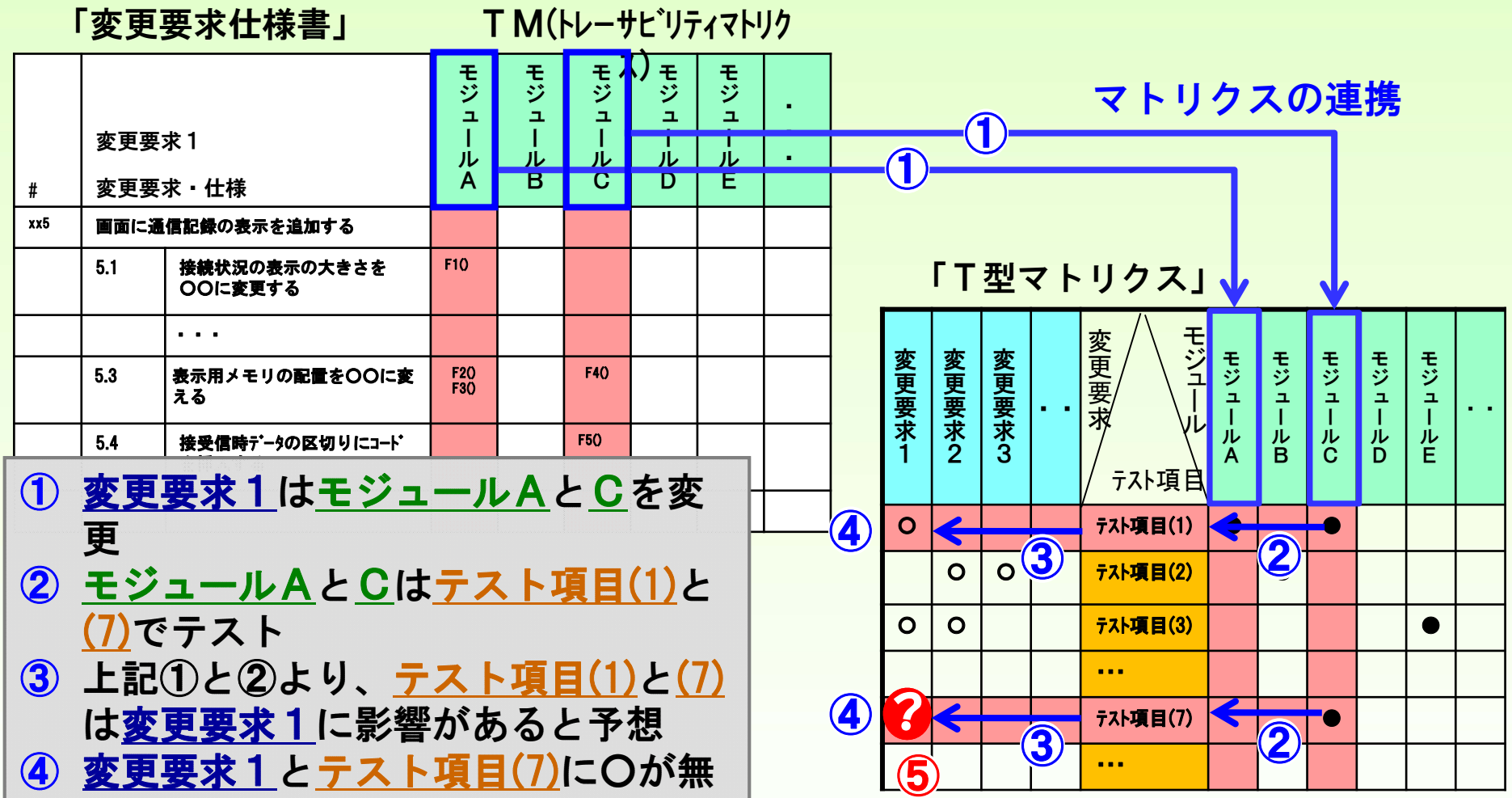
チェック

Ｔ型マトリクス

変更要求1	変更要求2	変更要求3	変更要求4	..	変更要求 ＼ モジュール テスト項目	モジュールA	モジュールB	モジュールC	モジュールD	モジュールE	..
○					テスト項目A	●		●			
	○	○			テスト項目B		●		●		
	○				テスト項目C		●			●	
					テスト項目D						
		○			テスト項目E	●				●	
			○		・・・				●		

整合性チェックの具体例（１／２）

- 「変更要求仕様書」→「Ｔ型マトリクス」



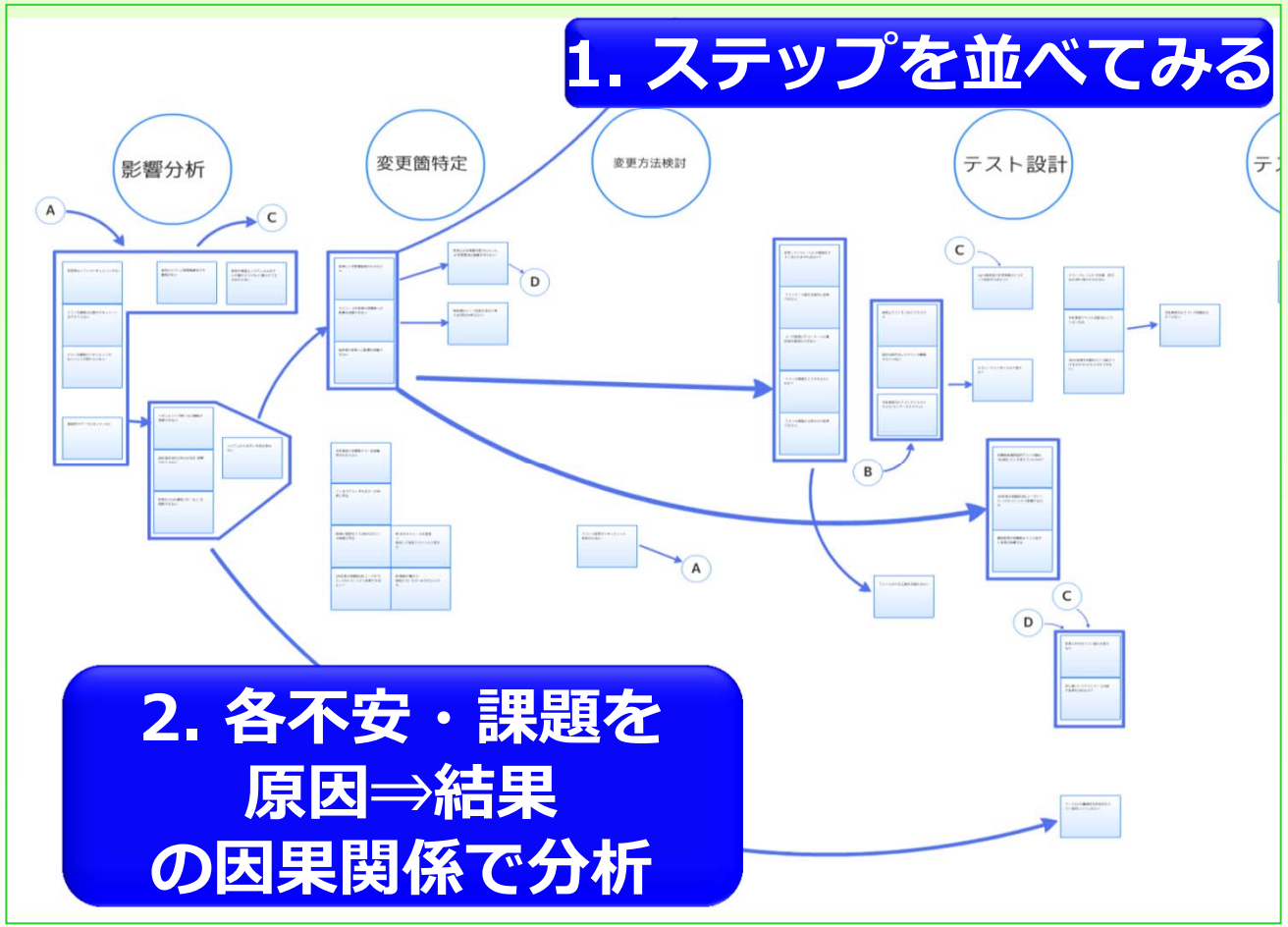
① 変更要求１はモジュールＡとＣを変更
② モジュールＡとＣはテスト項目(1)と(7)でテスト
③ 上記①と②より、テスト項目(1)と(7)は変更要求１に影響があると予想
④ 変更要求１とテスト項目(7)に〇が無いことが判明
⑤ テスト項目(7)に対する影響漏れまたは変更仕様漏れがないか確認

振り返り

- 既存のテスト項目全体を俯瞰しながら、変更要求との関連を明確にできるため、テスト方針に従ったテスト範囲を効率的に設定できることがわかった。
- ツールの考案に偏りすぎていたかもしれない
- もう一度、派生開発のテストの課題を議論していこう

現在までの活動

- 派生開発におけるソフトウェアテストの課題の洗い出し
- 課題の分析



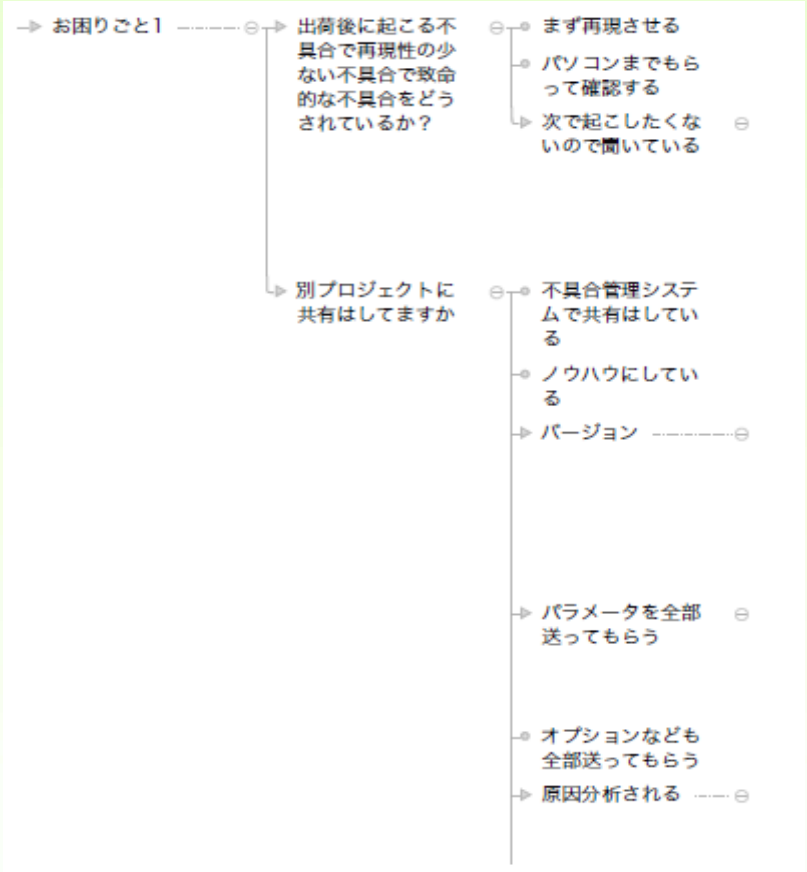
2016年 AFFORDD T4研究会 活動報告

派生開発推進協議会
T4　研究会

依田誠二

駆け込み寺

T04研究会のメンバーが抱えている課題を出し合うため駆け込み寺を実施



職場でのお悩みを打ち明けて、参加者から解決策・意見をもらう。

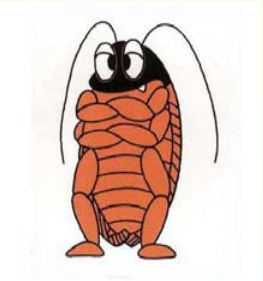
活動テーマを決定

お悩み：
ソフトウェアリリース後から1年以上経過してから市場から報告されるバグ（潜在バグ）が多い。
どの部署も同じ問題をかかえている。

テーマ：
潜在バグを減らすためにはどうすればよいか？

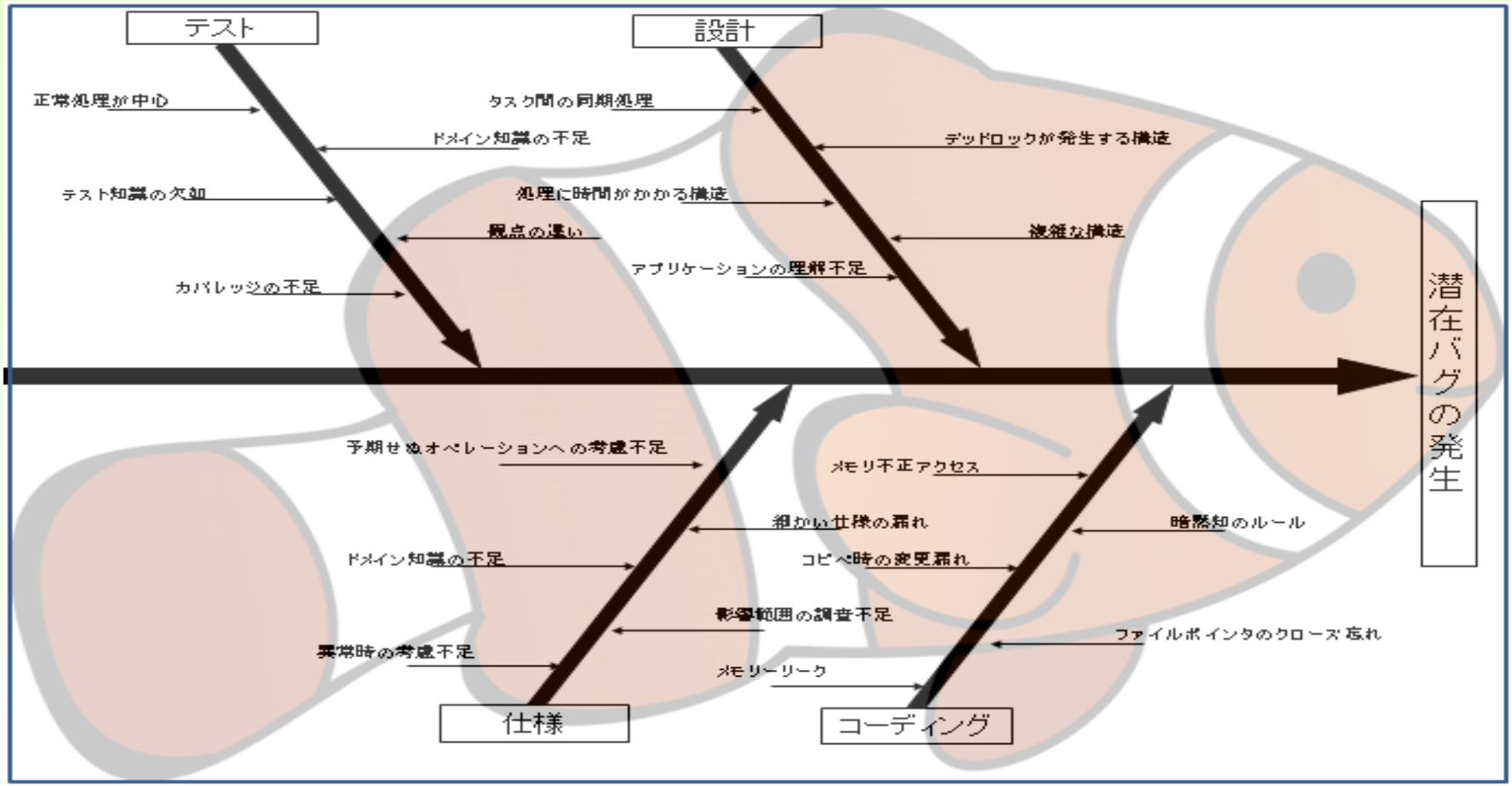
潜在バグの特徴

- 発生頻度が極端に低い
- カスタマーのオペレーションが変化することで発生する
- ハードウェアの故障に付随して発生する
- 時間の経過により発生する
- 原因がわからず根本的な対策が出来ない



要因分析

潜在バグが発生する要因を工程別に分析



仕様の問題点

- 影響範囲の調査不足
- 細かい仕様の漏れ
- 異常時の考慮不足
- 予期せぬオペレーションへの考慮不足
- ドメイン知識の不足

設計の問題点

- タスク間の同期処理
- 処理に時間がかかる構造
- アプリケーションの理解不足
- 複雑な構造
- デッドロックが発生する構造

コーディングの問題点

- メモリ不正アクセス
- 暗黙知のルール
- ファイルポインタのクローズ忘れ
- コピペ時の変更忘れ
- メモリーリーク

テストの問題点

- 正常処理が中心
- ドメイン知識の不足
- テスト知識の欠如
- カバレッジの不足
- 観点の違い

テストの問題点を深堀

- 観点の違いに着目
  - 正常系確認がテストの中心になっている。
  - カスタマーの使用方法を熟知していない。

過去不具合を活用することで上記の問題を解決出来ないか？

テスト手法

過去不具合を活用するテスト方法は幾つか提案されている

- スープカレー表
  - TEF北海道テスト勉強会(TEF-道)
- ソフトウェアテストにおけるバグ推測によるテスト設計手法の提案
  - ソフトウェア品質管理研究会 第31年度 第5分科会
- 故障事例によるテスト観点データベース構築とテスト設計への適用
  - ソフトウェア品質シンポジウム2012

結果

この手法を実際に使ってテストを行ってみる。

良かった点

- 細かい仕様でのバグが発見される。
- 機能の組み合わせで発生する不具合が発見される。

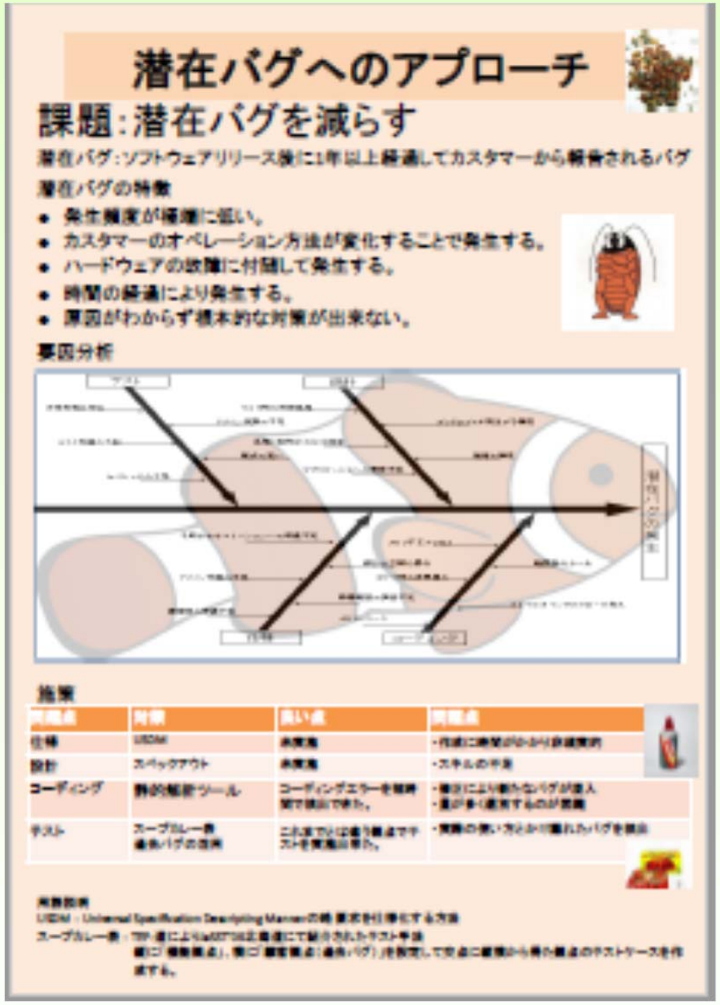
悪い点

- カスタマーの使い方から乖離したテストになる。
- 見つかったバグが影響が低いとして修正されずに終わってしまう。

期待する程の効果は得られなかった。

派生開発カンファレンス 2016

ポスター展示を実施して参加者に意見を聞いてみる。



USDMを使っているのでバグは出ないですよ。

バグが出やすいコードはリファクタしないとダメ

設計に時間をかけて開発行数を少なくする。

ウチもあるよ。解決策を教えて。

結局のところ開発時のバグが減れば潜在バグも無くなるのでは？

ソースコードの複雑度が高い部分をリファクタした

結局のところ開発時のバグが減れば潜在バグも無くなるのでは？

上流工程から改善しないとダメでは？

意見を工程ごとにまとめる

行程	意見
仕様	USDMを使って仕様決めに時間を費やせばリリース後のバグは出ない。
設計	設計段階で調査を行い少ない変更で開発するようにしている。
	バグを埋め込みやすい部分の設計を見直した。
コーディング	ソースコードの複雑な部分をリファクタした。
	静的解析ツールを使って指摘部分の修正を行った。
テスト	ソースコードの作りを意識して弱い部分に対してテストを実行したらどうか？
その他	結局ところ、開発時のバグが減れば潜在バグも無くなるにでは？
	開発の上流工程から改善しないとダメでは？
	ウチも同じ悩みをかかえているが、積極的なアプローチをかけれていない。

考察

仕様起因の不具合は開発段階の改善が必要

リリース済みソフトウェアの仕様は変更出来ない

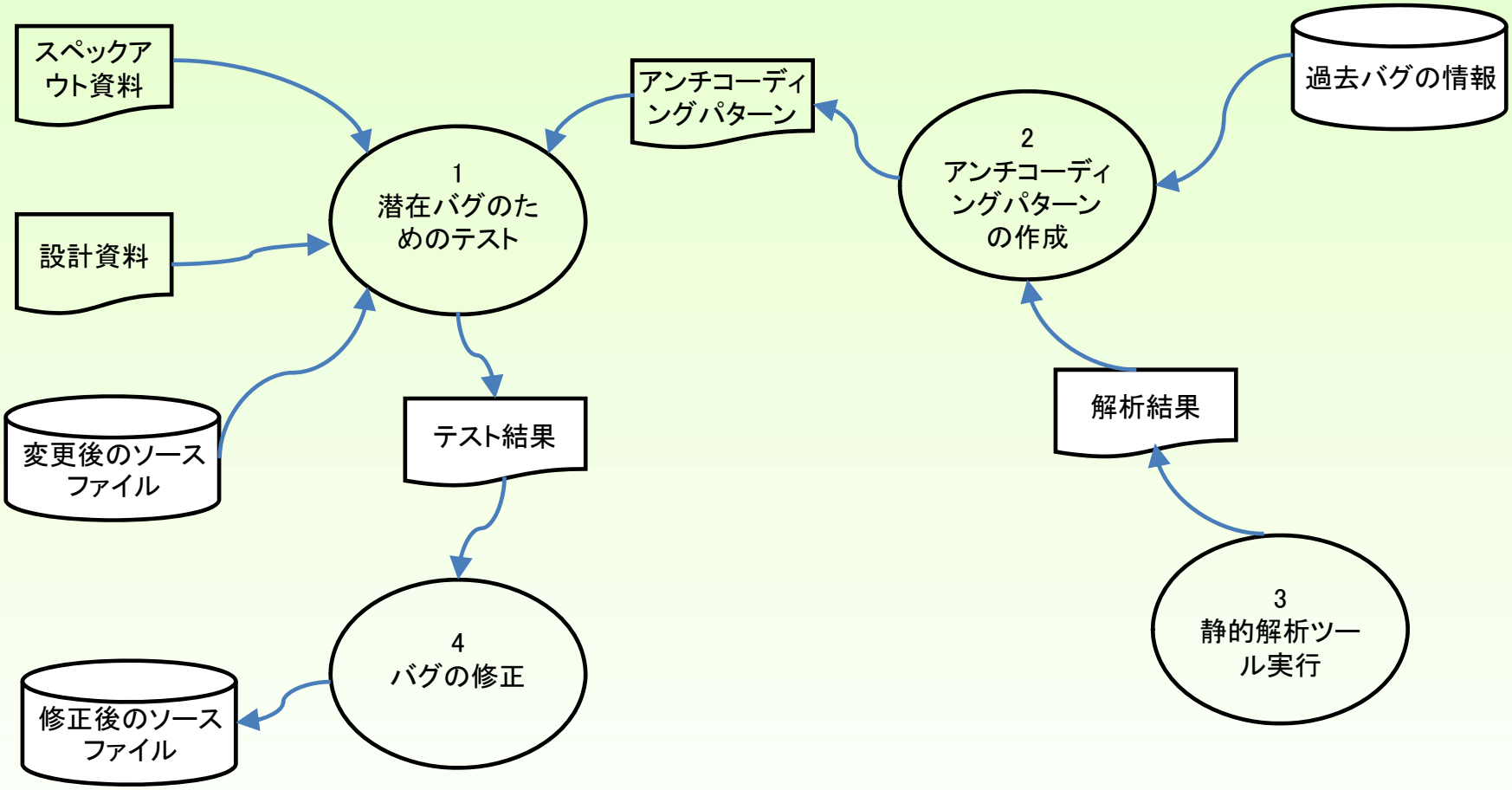
設計の複雑さが潜在バグを生み出している

リファクタも簡単に出来るわけではない。

テストで設計起因のバグを洗い出せないか？

これからの活動課題

- 設計起因バグをあぶりだすテストとは



T4研究会　研究員募集

派生開発のテストについて研究を行っています。

月に1度程度集まっています。

派生開発カンファレンスでの成果発表を目指します。

一緒にやりましょう。

(C) copyright 派生開発推進協議会