第 6 分科会（B グループ）

派生開発におけるモレ・ムダのないテスト設計

The test design process without overs and shorts in the derivative development

主　査	足立 久美	（株）デンソー		
副主査	奈良 隆正	NARAコンサルティング		
アドバイザ	清水 吉男	（株）システムクリエイツ		
研究員（リーダ）	奥山 剛	（株）山武		
（サブリーダ）	奥田 享一郎	TIS（株）、	吉本 吾朗	三菱UFJトラストシステム（株）
	永田 敦	ソニー（株)、	中森 博晃	パナソニック（株）
	村上 聡	日本信号（株)、	丸山 久	（株）クレスコ
	柳内 政宏	アルパイン（株）		

概要

派生開発で起きる代表的な問題点に、ソフトウェアのデグレードがある。デグレードとは、ソフトウェアの変更が影響し、変更前まで動いていた機能の品質が低下することである。派生開発におけるデグレードをテストで発見し防止するには、ソフトウェア変更の影響範囲を分析・特定し、適切なテスト設計を行う必要がある。影響範囲の分析・特定が特に重要であり、ここで「影響範囲の特定漏れにより必要なテストを漏らす」ことや、「影響範囲を絞り込めずに的を外したテスト（回帰テストを含む）を実施して余計なコストをかけてしまう」といったリスクを極力減らすことが課題となる。

本研究では、この課題の解決策の一つとして、「影響マトリクス」を利用したテスト設計プロセスの適用を提案する。提案する手法は、開発プロセスの変更と比べ導入が容易で、適用するシステムの種類を問わない。さらに、「影響マトリクス」は再利用性があり、設計レビューへの応用も可能である。本論文では手法の詳細を記載するとともに、実際の派生開発で発生した問題事例に対して手法を適用した場合のシミュレーション結果を報告する。

Abstract

In this paper, we describe the test design process without overs and shorts that utilized "Influence Matrix" in the derivative development.

One of the typical problems occur in the derivative development is the software degradation. The software degradation is that the software change causes the quality loss of the existing function.In order to discover and prevent it by tests in the derivative development, it’s necessary to identify and analyze the reach and impact of the software change, and design appropriate tests. Among the most important is to analyze the reach and impact of software change. Main challenge is to minimize the risks in this process, such as "to omit required test by insufficient impact assessment" and "to spend extra time and money for off-target tests （including regression testing）"

In this study, we suggest the adoption of the test design process utilized "Influence Matrix" as one of the solutions to them. Proposed method is easy to implement compared to the changes in the development process, regardless of the type of system to be applied.In addition, "Influence Matrix" has reusability and can be used for the design review, too. In this paper, we describe details of the method, and report on the result of the simulation that applied method to the problem case with actual derivative development.

1. 研究の背景

我々はSQIP研究会に所属し、派生開発プロセスの研究を行っている。特に派生開発に有効と思われる派生開発プロセスXDDPに着目し研究を開始した。XDDPの詳細は別に委ねるが、XDDPとは、

(c)Union of Japanese Scientists and Engineers(JUSE)

3点セット（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）を用いて限られた情報から最大限の効果を上げる開発プロセス手法である。[1]

XDDP の研究を進めるうち、我々は実際のプロジェクトに対してテストプロセスをどのように適用するかについての議論・文献が少ないことに気がついた。

XDDP では品質向上の手がかりとして、3点セット、スペックアウトドキュメントなどの付随ドキュメントが存在する。これらの限られた情報から、どのようにしたら XDDP を用いた派生開発で最適なテストができるか、漏れや無駄のないテスト設計をするにはどういったテストプロセスにすればよいか。

これを我々は研究テーマとすることにした。

2. 現状分析とその課題

我々はシステムテストや市場で発覚した複数の問題事例を分析することで、実際の派生開発プロジェクトで、テスト設計上の課題となるものを洗い出すことにした。

派生開発はすでに稼動しているシステムに新しい機能を組み込む、あるいは既存仕様を変更するプロセスである。稼動しているシステムを調査し、“ ここを変更すればよさそうだ ”と思って変更してみると、今回の変更と一見関係がないと考えられる機能がデグレードする問題が発生することがある。

その課題への対策を検討するために、各研究員が自社で発生した組込みシステムの派生開発における問題事例を持ち寄り、その要因を分析した。その分析結果の概要を「表 2-1：問題事例（一部）の分析結果」に示す。

問題事例を分析した結果では、稼動しているシステムを変更するにあたり、リソースの制約に対応出来ていないことが主な原因であることがわかった。

表 2-1：問題事例（一部）の分析結果

事例 No.	問題	変更点	原因	要因となったリソース	リソースの制約の分類
1	選択項目において、指定値と実際の設定値が異なる	機能を大幅に増やした	イベント処理スレッドが枯渇し、イベント処理順序が前後に入れ替わってしまった	メモリ	Heap メモリサイズ
2	データがずれてフラッシュメモリに記録されてしまう	機能追加	フラッシュメモリのファイル数が多いため、ファイルアクセスに時間がかかるようになり、データバッファがオーバフローを起こしデータが廃棄された	ファイルアクセス	バッファサイズ
3	外部メモリカードのデータが読み取れなくなる場合がある	メモリカードの種類追加	違う種類のメモリカードを同時に抜くと、ファイルシステムの多重起動で作業領域を壊し、メモリカード上のファイルシステム情報が破壊された	ファイルアクセス	排他

3. 解決策

リソース制約が原因となるデグレードを防止するためには、機能変更に対する影響範囲を特定し、その影響範囲に応じて適切なテストを実施する必要があると考えた。

そのためには、機能とリソースの関係を表現し、ある変更による影響範囲を分析するための資料が必要である。

XDDP では3点セットにより「What」「Where」「How」の視点から変更箇所や変更方法を記述することができるが、機能とリソース間の表現については「補助資料」を作成することを推奨している。ただし、その具体的なフォーマット、使い方、テストプロセスとの関係までは言及されていない。

そこで、我々は機能とリソースの関係を表現し、ある変更による影響範囲を分析する為の資料である「影響マトリクス」と、「影響マトリクス」を使用したテスト設計プロセスを考案した。

3.1. 影響マトリクス

影響マトリクスは、“ ソースファイルやモジュールなどの機能の単位 ”と“ ある変更により、別の機能に影響を及ぼす可能性のある要素 ”との関係を表すことで変更による影響範囲を表現するためのマ

(c)Union of Japanese Scientists and Engineers(JUSE)

トリクス表である。
影響マトリクスのフォーマットと使用方法を以下に記載する。

3.1.1. フォーマット

影響マトリクスのフォーマットを図 3-1 に示す。

影響因子 / モジュール	カテゴリA				カテゴリB				モジュール間関係			
	影響因子 A1	影響因子 A2	影響因子 A3	影響因子 A4	影響因子 B1	影響因子 B2	影響因子 B3	影響因子 B4	関係A	関係B	関係C	関係D
モジュールM1												
モジュールM2												
モジュールM3												
モジュールM4												
モジュールM5												

図 3-1：影響マトリクスのフォーマット

（1） 影響マトリクスの列項目

- カテゴリ：共通機能、共通リソース、モジュール間関係など、ある機能の変更により別の機能への影響を及ぼす可能性のある要素を分類したもの
- 影響因子：共通機能、共通リソースなどのカテゴリの中で、具体的な要素を記述する

（2） 影響マトリクスの行項目

XDDP の 3 点セットの 1 つである変更要求 TM（トレーサビリティ・マトリクス）の列要素を記載する。

変更要求TMの列要素と影響マトリクスを対応させることで、「変更仕様」⇒「変更箇所」⇒「変更による影響箇所」 の関連を表現する（図 3-2）。変更要求TMの列要素は、一般的にソースファイルやタスクなどの機能単位を配置する。[1]

変更要求TM

要求 / モジュール	モジュール M1	モジュール M2	モジュール M3	モジュール M4	モジュール M5
変更要求R1					
変更要求R2					

影響マトリクス

影響因子 / モジュール	カテゴリA				カテゴリB				モジュール間関係			
	影響因子 A1	影響因子 A2	影響因子 A3	影響因子 A4	影響因子 B1	影響因子 B2	影響因子 B3	影響因子 B4	関係A	関係B	関係C	関係D
モジュールM1												
モジュールM2												
モジュールM3												
モジュールM4												
モジュールM5												

図 3-2：変更要求 TM と影響マトリクスの関係

（3） 交点の表記法

影響マトリクスの交点は以下の表記法に従って記述する。

「○印」…行要素（機能）と影響因子の間で関連があると判断した箇所
「●印」…「○印」のついた箇所で、今回の変更対象となる箇所

「●印」のついたセルには、変更要求 TM との対応付けを示すために、変更要求 TM の交点を識別可能な情報を併せて記載する。（図 3-3 の I1～I4）

カテゴリとして、機能間の関係を示すものを設定する場合は、関係するモジュールを識別可能な情報を記載する。（図 3-3 の M1～M5）

(c)Union of Japanese Scientists and Engineers(JUSE)

3.1.2. 使用手順

影響マトリクスの使用手順を以下に記載する。

（1） 手順①：カテゴリ・影響因子の検討

対象システムの特性を考慮して、カテゴリ、影響因子を検討し、影響マトリクスを作成する。以前の開発や、別システムで使用したマトリクスが存在する場合は、既存のマトリクスに対する影響因子、カテゴリの追加・変更を行なう。

影響マトリクスは特定の組織内、またはシステムにおいて繰り返し使用し、使用の都度影響因子の見直しを行なっていくことでより効果的なものとなることが期待できる。

（2） 手順②：モジュールと影響因子の関連調査

流用元の仕様書またはソースコード等の調査（スペックアウト）を行いながら、調査対象のモジュールにおいて、影響因子との関連があると判断したら、影響マトリクスの交点に「○印」を付ける。ソースコード調査中に、影響因子として追加すべき項目が見つかった場合は適宜マトリクスに追加する。

（3） 手順③：影響箇所の特定（変更要求 TM との対応付け）

変更要求 TM において、変更が必要であると判断されたモジュールについて、影響マトリクスの影響因子との関連があるか（「○印」がついている列が存在するか）を確認する。「○印」がついている列が存在する場合、モジュールの変更内容による影響因子と、関連部分への影響を評価する。影響ありと判断した場合、「●印」に変更するとともに、変更要求 TM の交点情報を識別可能な情報を付記する。

（4） 手順④：影響範囲の分析

影響マトリクスから、変更による影響範囲の分析を行なう。

3.1.3. 記述例

手順①～④に従って作成した影響マトリクスの記述例を図 3-3 に示す。

図 3-3 のマトリクスにおけるカテゴリ､影響因子は各研究員が持ち寄った事例をもとに検討したものである。

影響因子 / モジュール	共通機能				リソース				モジュール間関係			
	ファイルアクセス	通信機能	タイマ	共有データ管理	グローバル変数	メモリ	FIFO	セマフォ	タイミング	順序	排他	依存
モジュールM1	●：I1						○		M2		M4	
モジュールM2			●：I2		○				M1			
モジュールM3		○				●：I4					M5	
モジュールM4	○				●：I3							
モジュールM5				○		○						M2

「●印」に付記した I1～I4 は、変更要求 TM の交点を示す ID を表す

図 3-3：影響マトリクスの記述例

3.2. プロセス

XDDPにおける影響マトリクスの位置付けを理解しやすくするために、XDDPの変更プロセスのPFD（Process Flow Diagram）[1]に、影響マトリクスを組み込んだものを、図 3-4 に示す。図中の  はプロセスを、  は成果物を示す。プロセス中の数字はプロセス番号を表す。組み込んだ影響マトリクスを  で示す。

(c)Union of Japanese Scientists and Engineers(JUSE)

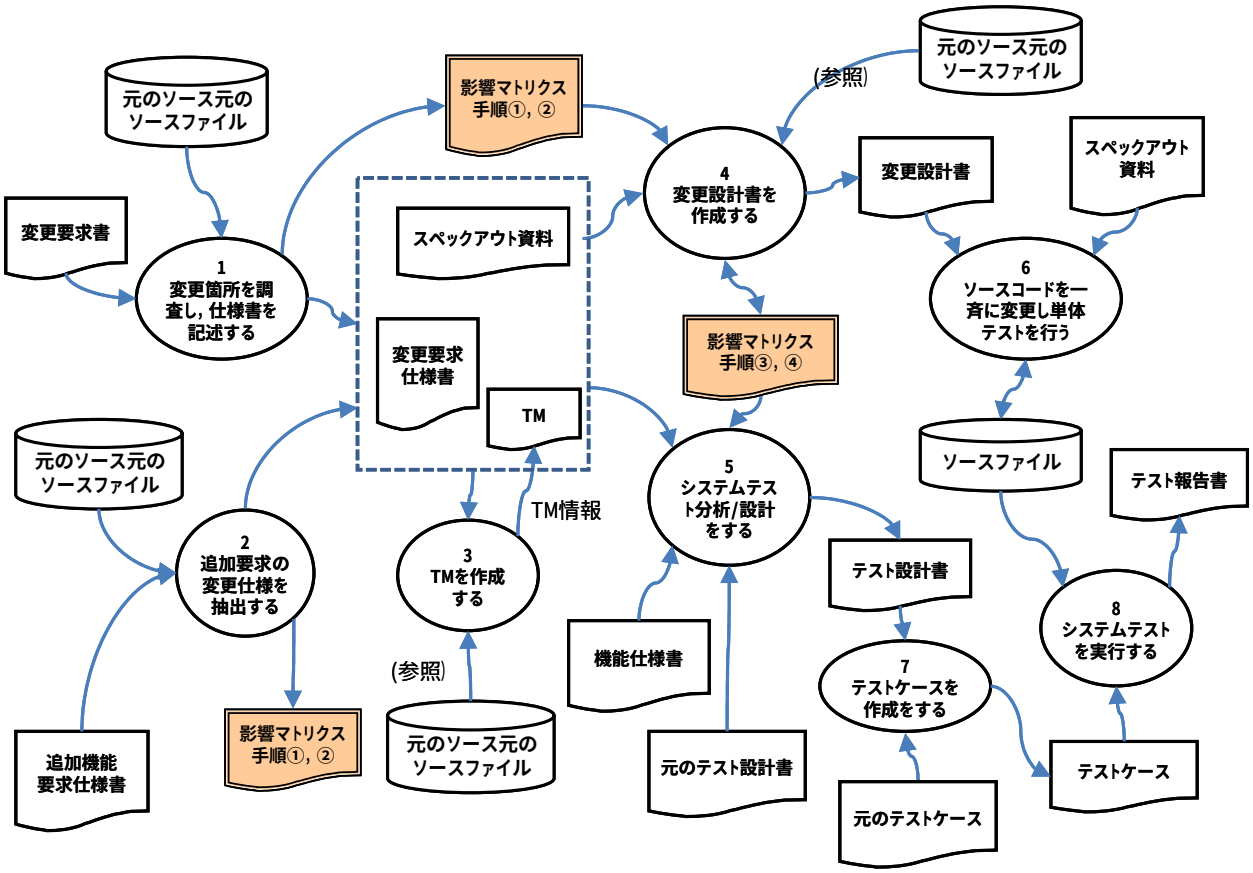


図 3-4：影響マトリクスの位置づけ

以降に示す各図（図 3-5～3-7）は、変更設計書およびシステムテスト仕様書を作るところまでのプロセスをあらわしている。

【プロセス 1：変更箇所を調査し、仕様書を記述する】

プロセス 1 の下位プロセスを図 3-5 に示す。まず始めに、影響マトリクスの手順①、②を実行していく。プロセス 1.1 において、スペックアウトの際に発見した影響要素を洗い出しておく。

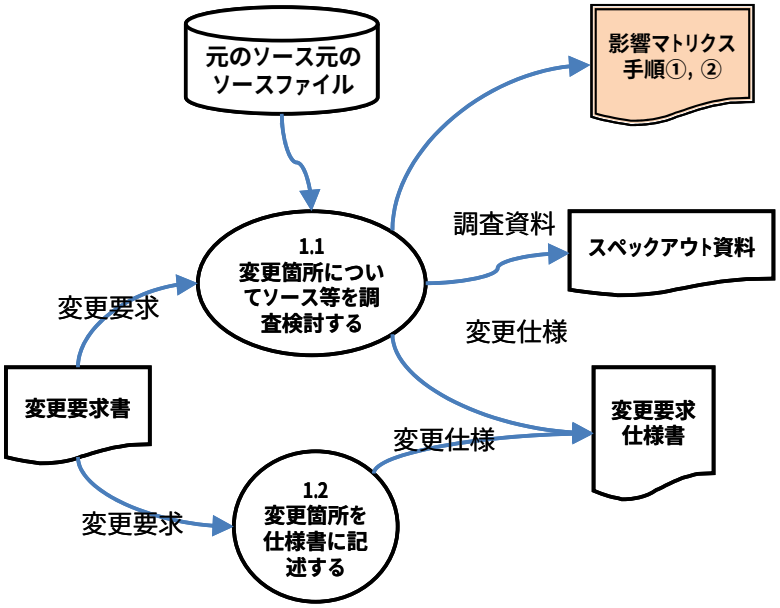


図 3-5：プロセス 1 の下位プロセス

(c)Union of Japanese Scientists and Engineers(JUSE)

【プロセス 2：追加要求の変更仕様を抽出する】

次にプロセス 2 でも同様に影響マトリクスの手順①、②を実行していき、共有しているリソースの洗い出しを行っておく。

【プロセス 4：変更設計書を作成する】

プロセス 4 の下位プロセスを図 3-6 に示す。まず始めに、従来のように「4.1：変更要求仕様書、TM から変更設計書を書き出す」を行った後、「4.2：影響するリソースの情報を影響マトリクスに書き出す」によって影響マトリクスの手順③、④ を行う。次に、「4.3：変更内容に応じた単体テスト項目を変更設計書に書き出す」によって単体テストの仕様を確認項目として変更設計書に記述する。次に「4.4：共有リソースに対する単体テスト項目を変更設計書に書き出す」というプロセスで、4.2 で作られた影響マトリクスを参照して共有リソースで競合する項目で単体テストを行うべきものを変更設計書に追加していく。このプロセスでは、共有リソースの競合という観点から、システムテストレベルでは実現できないテストを単体テストで実現することを考える。

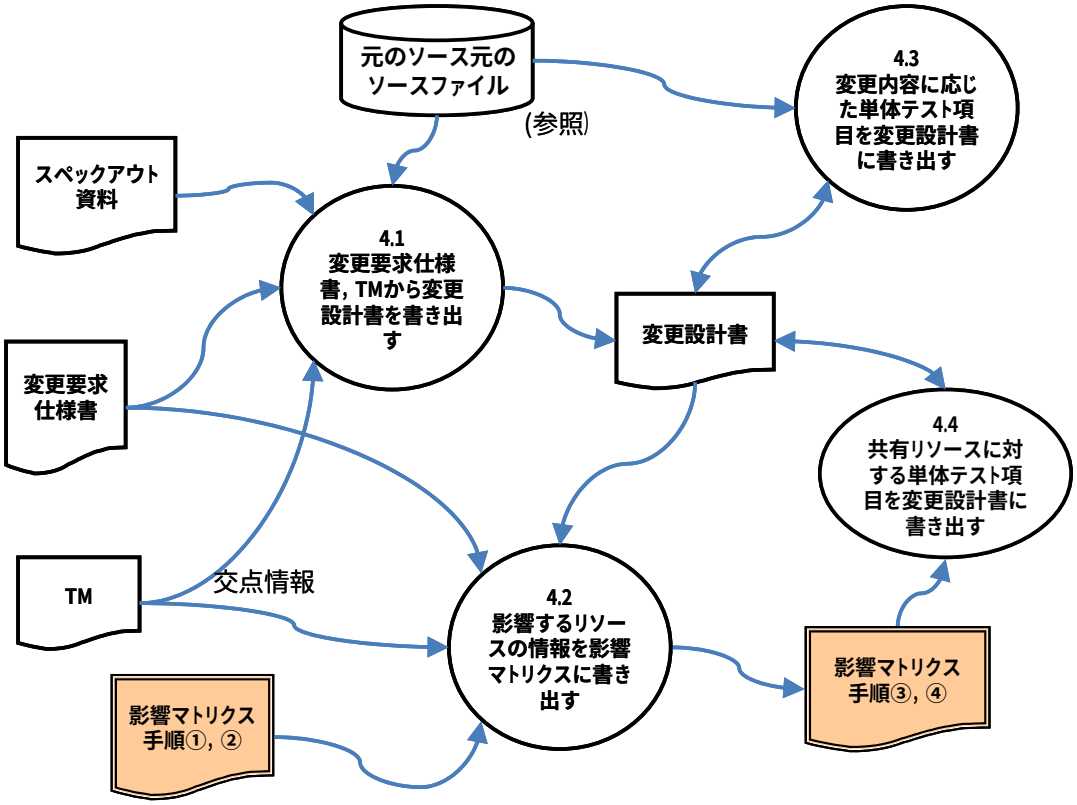


図 3-6：プロセス 4 の下位プロセス

【プロセス 5：システムテスト分析/設計をする】

プロセス 5 の下位プロセスを図 3-7 に示す。ここでは、システムテストのテスト分析と設計のプロセスの例を示している。実際は、ドメインや製品の規模、テストポリシーによって、プロセスや成果物は変わる。ここでは、代表的な例を示し、影響マトリクスの位置づけを示している。「5.3：リソースの競合に対するテスト項目を洗い出す」のプロセスにおいて、影響マトリクスによって導かれる観点からテスト設計を行う。また、「5.4：回帰テストの項目を洗い出す」においても影響マトリクスは、テストの範囲を決める際の参考となる。

(c)Union of Japanese Scientists and Engineers(JUSE)

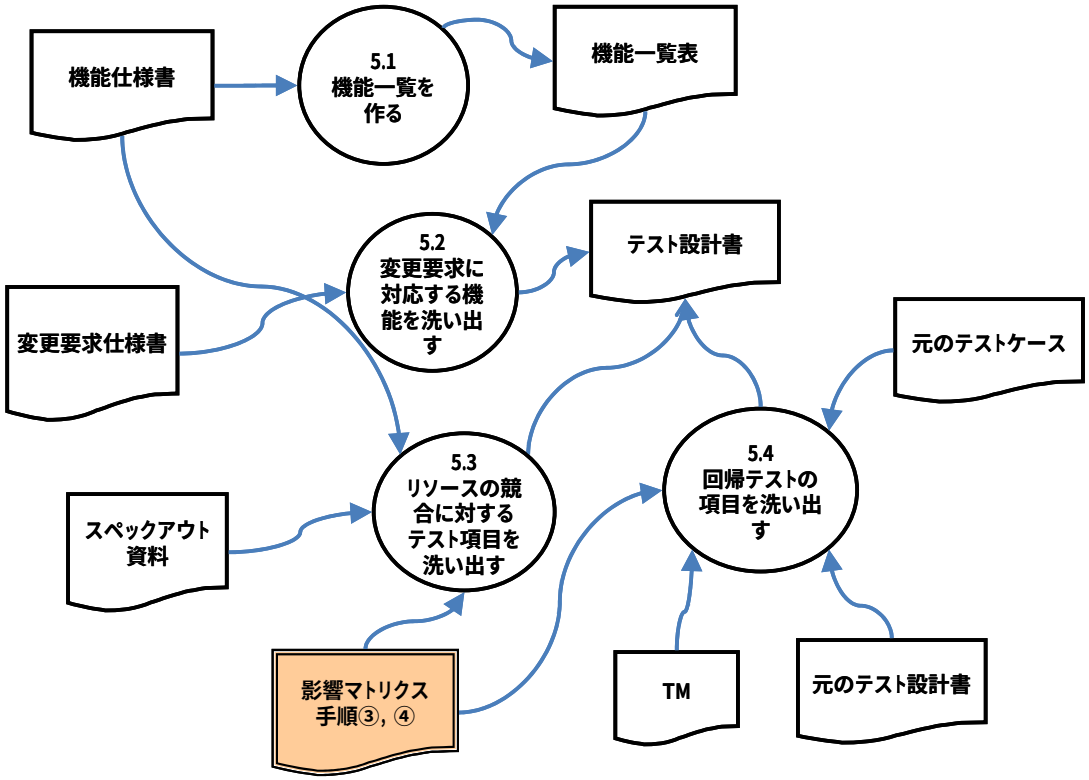


図 3-7：プロセス 5 の下位プロセス

このように、影響マトリクスの適用範囲はシステムテストに留まらない。リソースの競合に対するテストを行う場合は、その競合の性質によって単体テストレベルもしくは結合テスト、システムテストレベルでテスト設計をする際に、影響マトリクスを参照して必要なテストを追加していく。つまり、影響マトリクスをさまざまなテストレベルにおいて共有することで、リソース競合を漏れなく検証することができ、そして一方で、同じ観点での違うレベルのテストどうしのダブりを防ぐことができる。

4. 解決策の検証

4.1. 検証内容

現状分析に用いた 3 事例に対し、影響マトリクスを使用したテスト設計プロセスのシミュレーションを行った。シミュレーションにより、影響マトリクスの使用は事例で発生したような問題を検出するためのテスト設計に寄与するのか、あるいは寄与しないのかの評価を行った。その結果は、テストでの検出への寄与が見込まれるというものであった。

次に、影響マトリクスの使用によるテスト設計への寄与は、前述の 3 事例以外の問題事例においても確認出来るか評価するために、追加で 7 事例に対してシミュレーションを行った。

計 10 事例に対してシミュレーションを行った結果、7 事例でテストでの検出への寄与が見込まれ、3 事例では寄与が見込めないというものであった。寄与が見込めなかった 3 事例については、「4.2 検証結果」にて分析結果を示す。

シミュレーションの詳細は、「付録 A，表 A-1～4：問題事例への影響マトリクスの使用シミュレーション結果」に記載する。

以下に事例 No.1 を例に、影響マトリクスの使用イメージを示す。

(1)　手順①：カテゴリ・影響因子の検討

対象システムの特性を考慮して、カテゴリ、影響因子を検討し、影響マトリクスの列を作成する。

影響マトリクスの行要素は、トレーサビリティマトリクスの列要素から作成する。

(c)Union of Japanese Scientists and Engineers(JUSE)

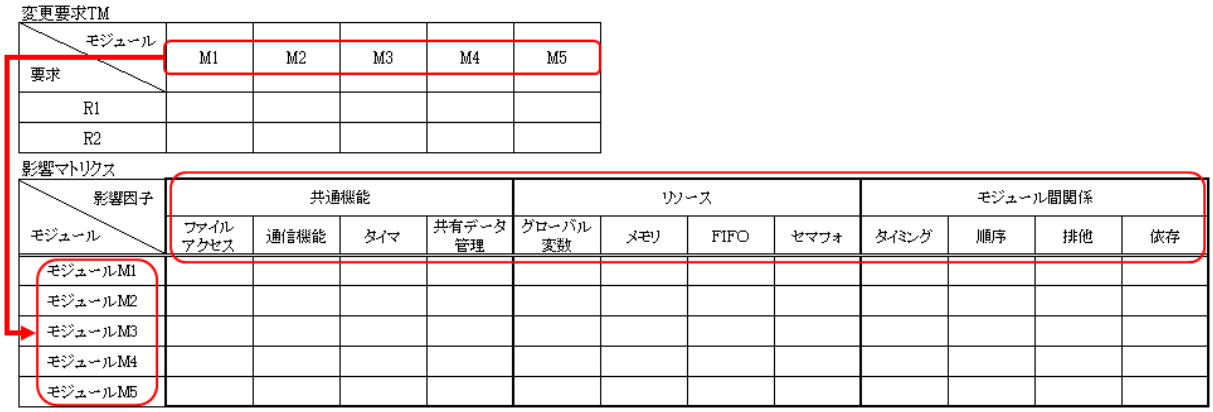


図 4-1：影響マトリクスの使用イメージ（1/4）

（2） 手順②：機能と影響因子の関連調査

ソースコードの調査を行いながら、影響因子との関連をチェックする。

例えば、モジュール M1、M2、M3 が Heap メモリを使用している（獲得や返却など）ことを識別したら、モジュール M1、M2、M3 のメモリ欄を「○」を記述する。

影響因子 / モジュール	共通機能				リソース				モジュール間関係			
	ファイルアクセス	通信機能	タイマ	共有データ管理	グローバル変数	メモリ	FIFO	セマフォ	タイミング	順序	排他	依存
モジュールM1	○			○		○	○			M1		
モジュールM2	○		○	○		○	○			M2		
モジュールM3		○				○						
モジュールM4			○				○	○			M5	
モジュールM5	○						○	○			M4	

図 4-2：影響マトリクスの使用イメージ（2/4）

（3） 手順③：影響箇所の特定（変更要求 TM との対応付け）

変更要求 TM において、変更が必要であると判断されたモジュールについて、影響マトリクスの影響因子との関連があるか（「○印」がついている列が存在するか）を確認する。

例えば、モジュール M3 内の変更 I1 と、Heap メモリとの関わりをレビューする。変更 I1 が Heap メモリを使用している事を認識し「●」する。

変更要求TM

モジュール / 要求	M1	M2	M3	M4	M5
R1			I1		
R2				I2	

影響マトリクス

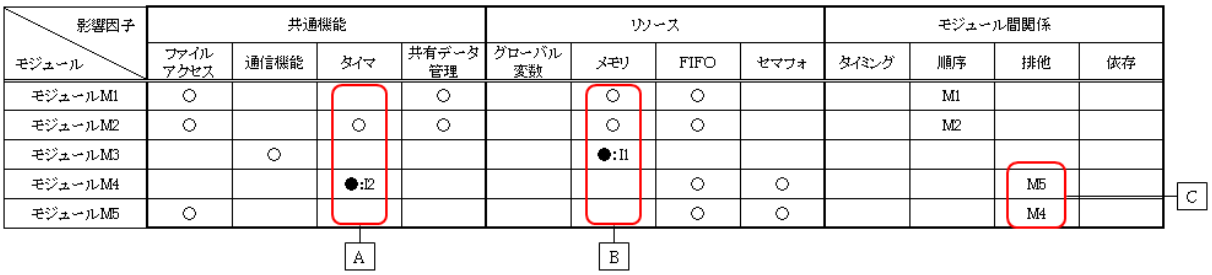
影響因子 / モジュール	共通機能				リソース				モジュール間関係			
	ファイルアクセス	通信機能	タイマ	共有データ管理	グローバル変数	メモリ	FIFO	セマフォ	タイミング	順序	排他	依存
モジュールM1	○			○		○	○			M1		
モジュールM2	○		○	○		○	○			M2		
モジュールM3		○				●:I1						
モジュールM4			●:I2				○	○			M5	
モジュールM5	○						○	○			M4	

図 4-3：影響マトリクスの使用イメージ（3/4）

（4） 手順④：影響範囲の分析

変更方法（変更設計書）と影響範囲（影響マトリクス）をもとにテストケースの設計を行なう。

(c)Union of Japanese Scientists and Engineers(JUSE)



A … モジュール M4 への変更 I2 により、
タイマリソースを介してモジュール M2 に影響がある可能性がある。

B … モジュール M3 への変更 I1 により、
メモリリソースを介してモジュール M1、M2 に影響がある可能性がある。

C … モジュール M4 は今回変更対象なので、
モジュール間の排他で関連するモジュール M5 に影響がある可能性がある。

図 4-4：影響マトリクスの使用イメージ（4/4）

影響マトリクスの使用により、上記 A、B、C の影響範囲が示される。
例えば、影響範囲 B から次のような観点でのテストケースの設計の必要性が示唆される。

観点 ： “ I1 の変更点によってもたらされたメモリに対する変化点が
モジュール M1、M2 に与えている影響が期待通りであるか ”

その結果、ストレス状況下でのメモリ使用状況が想定範囲内であるか確認するようなテストケースが設計されれば、事例 No.1 の原因である “ イベント処理スレッドの枯渇 ” が、テスト段階で顕在化する可能性が向上する。

4.2. 検証結果

本来は、我々の提案施策を、実稼動プロジェクトにおいて運用し効果を検証するべきであるが、今回は時間的な都合から実施を断念し、研究員が持ち寄った問題に対して、提案施策をあてはめての、シミュレーションによる検証となった。

研究員が持ち寄った数十件の不具合事例のうち、組込みシステムに関する不具合から、ランダムに 10 件の事例をピックアップし、我々の提案施策を適用した。

その結果、10 件中 7 件において、影響マトリクスを作成することによる効果が見込まれる結果が得られた。

不具合の検出が見込める事例を分析してみた所、メモリやファイルへの「物理リソースへのアクセス競合」や「共通機能のリソース使用競合」（タイマ、FIFO、等）などに有効であることが分かった。

我々の考案した影響マトリクスは、特定のリソースを、複数のモジュールやソースファイルで共同利用するソフトウェアのテスト設計時に、リソースの制約が問題要因となる不具合を効果的に検出するテストケース設計の支援をすることができる。したがって、影響マトリクスは、XDDP を用いた派生開発におけるあらゆるテストレベルのテスト設計に対して有効な手法であると言える。

不具合の検出が見込めなかった事例については、“ そもそも処理が漏れていた ” というものと、“ CPU 負荷が原因による処理遅延 ” の不具合であった。前者に関しては、「リソース」に関連しない不具合であり影響マトリクスの狙いから外れるものである。XDDP を忠実に行うことで防止してほしい不具合である。後者の原因は、今回のシミュレーションで作成した影響マトリクスの影響因子に「CPU」が含まれていなかったことにある。

(c)Union of Japanese Scientists and Engineers(JUSE)

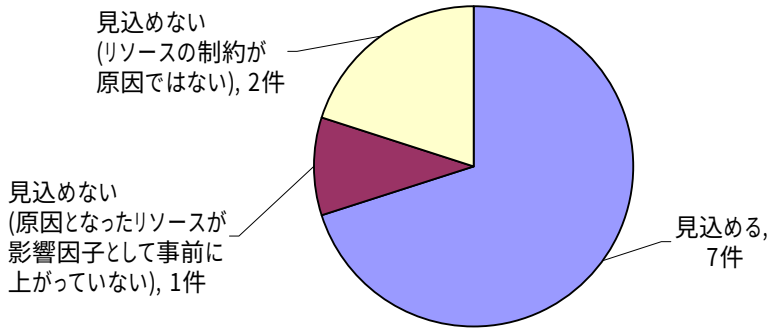


図 4-5：テスト設計への寄与の評価結果

影響因子は、ソフトウェアの対象システム（組込み、エンタープライズ）だけでなくドメイン（自動車制御、携帯電話端末、家電製品、…）や、レイヤ（ドライバ、通信、アプリケーション、…）によって、異なってくるものである。そのため、今回の提案で影響因子を既定することはしていない。PDCA を回しながらフィードバックし、その開発現場に合う項目を確立し、影響マトリクスの精度を高めていく必要がある。

5. 結論

5.1. 考察

実際の開発現場では、影響マトリクスと同等の内容、つまり変更点から変化点を追跡すること、を既に行っているかも知れない。しかし、それは担当者の暗黙知となっていることが多い。本論文では、この知識を影響マトリクスとして、明文化した。

影響マトリクスは、知識を蓄積していくためのツールとして作用するため、特に経験の浅い担当者に対して、効果を発揮すると我々は考える。経験豊かな担当者に対しても、うっかりミスの防止に有効であろう。さらに、効果が期待できる検証で重要となるポイントとして、テスト担当者にも明示されるなど、設計者とテスト担当者のブリッジを果たすことがある。

5.1.1. 応用例

影響マトリクスは、テスト以外への応用も可能である。

（1） 設計レビューでの活用

影響マトリクスを設計レビューの対象資料とする。これにより、関連機能への影響の見落としが無いか確認できる。

（2） エンタープライズシステムへの適用

影響マトリクスを利用したテスト設計プロセスは、対象システムを限定するものではない。影響因子を適切に選定すれば、エンタープライズシステムへの適用も可能である。

エンタープライズシステムでは、開発言語に Java などのオブジェクト指向言語を用いることが多い。これにより、組込みシステムと比較して、独立性の高いモジュールを作成し易い環境にある。よって、Web システムなどでは、影響マトリクスの恩恵は少ないかも知れない。しかし、スレッドプログラミングの分野では、メモリ空間が共有されるため、組込みシステムと同様に、モジュールの依存度が高くなる。よって、エンタープライズシステム、特にスレッドプログラミングの分野においても有効であると考える。

5.1.2. 組織への導入

新しいプロセスの導入は、常に障壁を伴うが、影響マトリクスの導入は比較的容易である。なぜならば、影響マトリクスの内容は、ソースコードの調査時にメモ、または担当者の頭の中に残されている情報である。よって、担当者の作業負荷を増やすものではないからである。

(c)Union of Japanese Scientists and Engineers(JUSE)

ただし、影響マトリクスの実際の運用においては、影響因子の数が増え過ぎないようにする必要がある。影響因子が増えすぎると、影響マトリクスは、形骸化して、役に立たなくなるため、定期的に見直しを行う必要がある。

5.1.3. 今後の課題

実際の開発プロジェクトに影響マトリクスを適用し、効果を測定したい。

5.2. まとめ

今回、XDDP のプロセスと成果物をもとに、本研究で考案した影響マトリクスとそれを使った追加プロセスを実施することが、XDDP を用いた開発のテスト設計において有効であることを確認することができた。

本研究では、影響マトリクスの影響因子をそれぞれの研究員の経験をもとに定義した。今後より多くのプロジェクトで適用されることで、適用した組織またはプロジェクトにとって、有効な観点や経験を蓄積することにより、更に精度の高い影響マトリクスが作成され、派生開発におけるテスト設計が高いレベルで実現されていくことを期待したい。

6. 参考文献

[1] 清水 吉男：「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、2007
[2] 清水 吉男：要求を仕様化する技術・表現する技術、技術評論社、2005
[3] 清水 吉男：「失敗しない派生開発」, Software People, Vol.8, 技術評論社、2006
[4] 清水 吉男：「要求の仕様化入門」 , Software People, Vol.4, 技術評論社、2004
[5] 松本 吉弘：ソフトウェアエンジニアリング基礎知識体系 - SWEBOK 2004 -、オーム社、2005
[6] 松本 吉弘：ソフトウェア開発への SWEBOK の適用、オーム社、2005
[7] SQuBOK 策定部会：ソフトウェア品質知識体系ガイド －SQuBOK Guide－、オーム社、2007
[8] ISTQB：テスト技術者資格制度 Advanced level シラバス 日本語版 Version 2007.J01、JSTQB、2007
[9] ISTQB：テスト技術者資格制度 Foundation level シラバス 日本語版 Version 2007.J01、JSTQB、2007
[10] ISTQB：ソフトウェアテスト標準用語集（日本語版）Version 2.0.J01（2007 年 12 月 02 日）、JSTQB、2007

(c)Union of Japanese Scientists and Engineers(JUSE)

付録 A　影響マトリクスの使用シミュレーション結果

表 A-1：問題事例への影響マトリクスの使用シミュレーション結果（1/4）

事例 No.	問題事例					影響マトリクスの使用手順 （『手順①カテゴリ・影響因子の検討』は各事例の共通作業として事前実施済み）					影響マトリクスの使用は、問題を検出するためのテスト設計への寄与が見込める？／見込めない？
	現象	変更点	原因	制約となるリソース	制約の分類	手順② 機能と影響因子の関連調査	手順③ 影響箇所の特定（変更要求 TM との対応付け）	手順④ 影響範囲の分析	影響範囲の分析のポイント	影響因子	
1	選択項目において、指定値と実際の設定値が異なる	機能を大幅に増やした	イベント処理スレッドが枯渇し、イベント処理順序が前後に入れ替わってしまった	メモリ	Heap メモリサイズ	調査過程において、Heap メモリを使用している（獲得や返却など）モジュールを識別したら、モジュールのメモリ欄を『○』を記述する。	モジュール内の変更箇所と、Heap メモリとの関わりをレビューする。変更箇所が Heap メモリを使用している事を認識し『●』にする。	『●』をチェックしたモジュールの変更点によってもたらされたメモリに対する変化点が、『○』のモジュールに与えている影響が期待通りであることを確認するテスト設計を行う。 ・ストレス状況下でのメモリ使用状況が想定範囲内であるか など	非機能テスト（ストレステスト）の必要性を見出せることがポイントとなる	メモリ	見込める
2	データがずれてフラッシュメモリに記録されてしまう	機能追加	フラッシュメモリのファイル数が多いため、ファイルアクセスに時間がかかるようになり、データバッファがオーバフローを起こしデータが廃棄された	ファイルアクセス	バッファサイズ	調査過程において、ファイルシステムを使用しているモジュールを識別したら、モジュールのファイルアクセス欄を『○』を記述する。	モジュール内の変更箇所と、ファイルシステムとの関わりをレビューする。変更箇所がファイルアクセスしていることを認識し『●』にする。	『●』をチェックしたモジュールの変更点によってもたらされたファイルアクセスに対する変化点が、『○』のモジュールに与えている影響が期待通りであることを確認するテスト設計を行う。 ・ストレス状況下でのファイル作成（限界値が SLA に適合するか？限界を超えたときの例外処理は期待通りか？ など	非機能テスト（ストレステスト）の必要性を見出せることがポイントとなる	ファイルアクセス	見込める
3	外部メモリカードのデータが読み取れなくなる場合がある	メモリカードの種類追加	違う種類のメモリカードを同時に抜くと、ファイルシステムの多重起動で作業領域を壊し、メモリカード上のファイルシステム情報が破壊された	ファイルアクセス	排他	調査過程において、ファイルシステムを使用しているモジュールを識別したら、モジュールのファイルアクセス欄を『○』を記述する。	モジュール内の変更箇所と、ファイルシステムとの関わりをレビューする。変更箇所がファイルアクセスしていることを認識し『●』にする。	『●』をチェックしたモジュールの変更点によってもたらされたファイルアクセスに対する変化点が、『○』のモジュールに与えている影響が期待通りであることを確認するテスト設計を行う。 ・ファイルシステムの使用が競合したとき（追加したメモリカードを含む挿抜の競合など）の振る舞いの確認 など	非機能テスト（ストレステスト）の必要性を見出せることがポイントとなる	ファイルアクセス	見込める

表 A-2：問題事例への影響マトリクスの使用シミュレーション結果（2/4）

事例 No.	問題事例					影響マトリクスの使用手順 （『手順①カテゴリ・影響因子の検討』は各事例の共通作業として事前実施済み）					影響マトリクスの使用は、問題を検出するためのテスト設計への寄与が見込める？／見込めない？
	現象	変更点	原因	制約となるリソース	制約の分類	手順② 機能と影響因子の関連調査	手順③ 影響箇所の特定（変更要求 TM との対応付け）	手順④ 影響範囲の分析	影響範囲の分析のポイント	影響因子	
4	リモコンの電源ボタンが利かなくなる	機能追加	通信におけるタイマ ID が，別のモジュールのタイマの ID と競合し、別モジュールはそのタイマを解除してしまった	タイマ	排他	調査過程において、タイマを使用しているモジュールを識別したら、モジュールのタイマ欄を『○』を記述する。	モジュール内の変更箇所と、タイマとの関わりをレビューする。変更箇所がタイマを使用していることを認識し『●』にする。	『●』をチェックしたモジュールの変更点によってもたらされたタイマに対する変化点が、『○』のモジュールに与えている影響が期待通りであることを確認するテスト設計を行う。 ・タイマリソースを利用している機能のリグレッション など	回帰テストの必要性を見出せることがポイントとなる	タイマ	見込める
5	ある通信機器内で通信エラーが発生し、通信が途絶えてオフライン動作に切り替わった	プロセスの追加	プロセス間通信で使用しているメッセージキュー使用率が 100%となった。そして「待ち状態」になったが、実装は、キューの使用割合がそこまであがることを想定していなかった	FIFO	キューサイズ	調査過程において、メッセージキュー（FIFO）を使用しているモジュールを識別したら、モジュールの FIFO 欄を『○』を記述する。	モジュール内の変更箇所と、FIFO との関わりをレビューする。変更箇所が FIFO を使用している事を認識し『●』にする。	『●』をチェックしたモジュールの変更点によってもたらされた FIFO の変化点が、『○』のモジュールに与えている影響が期待通りであることを確認するテスト設計を行う。 ・ストレス状況下での FIFO の使用状況が想定範囲内であるかなど	非機能テスト（ストレステスト）の必要性を見出せることがポイントとなる	FIFO	見込める
6	通信において一時的に異常が発生（自動回復）していることが判明した。	機能追加	電文管理処理を一元処理しているブロックが高負荷となり、定期電文のタイムアウトが発生し通信が切断された	CPU	負荷	調査過程において、メッセージキュー（FIFO）を使用しているモジュールを識別したら、モジュールの FIFO 欄を『○』を記述する。	モジュール内の変更箇所と、FIFO との関わりをレビューする。変更箇所が FIFO を使用している事を認識し『●』にする。	『●』をチェックしたモジュールの変更点によってもたらされた FIFO の変化点が、『○』のモジュールに与えている影響が期待通りであることを確認するテスト設計を行う。 ・ストレス状況下での FIFO の使用状況が想定範囲内であるかなど	FIFO のストレス試験の過程で CPU リソース不足が発生する可能性はあるが、FIFO のストレス試験のみで CPU の高負荷が具現化するとは限らない。仮に発生し、問題を認識したとしても、それは偶然である。		見込めない

表 A-3：問題事例への影響マトリクスの使用シミュレーション結果（3/4）

事例No.	問題事例					影響マトリクスの使用手順 （『手順①カテゴリ・影響因子の検討』は各事例の共通作業として事前実施済み）					影響マトリクスの使用は、問題を検出するためのテスト設計への寄与が見込める？／見込めない？
	現象	変更点	原因	制約となるリソース	制約の分類	手順② 機能と影響因子の関連調査	手順③ 影響箇所の特定（変更要求 TM との対応付け）	手順④ 影響範囲の分析	影響範囲の分析のポイント	影響因子	
7	待機 2 重系装置のシステムにおいて、切換時に主系、従系で通信するメッセージが遅れて送信される。	機能追加	ソフトウェア流用時に、流用すべきソフトが一部漏れており、適正なタイミングでメッセージ送信行うことをチェックする機能が働いていなかった。						既存機能のデグレでなく、そもそも追加した機能が動いていない。 XDDP の適用（変更設計書の記載事項に基づく試験）で検出すべきと考える。	FIFO	見込めない
8	地図上の目的地の検索時、誤った場所が目的地になってしまう。	年次更新で DB のあるデータ項目の桁数が全て 10 桁になった。（今までは、9 桁と 10 桁が混在）	データ項目の桁数が、9 桁と 10 桁が混在している時のみに適用すべきパッチの外し漏れ。	DB	共有	調査過程において、DB を使用しているモジュールを識別したら、モジュールの共有データ管理欄を『○』を記述する。	モジュール内の変更箇所と、DB の更新との関わりをレビューする。変更箇所が DB 更新の影響を受けることを認識し『●』にする。	『●』をチェックしたモジュールの変更点によってもたらされた DB の変化点が、『○』のモジュールに与えている影響が期待通りであることを確認するテスト設計を行う。 ・DB 更新前後の振る舞いが互換であるか など	回帰テストの必要性を見出せることがポイントとなる	共有データ管理	見込める
9	音声認識で音楽 DISC のトラック NO を発話しても指定した曲が再生されない。	BUS コマンドの変更	BUS コマンドの変更への、アプリケーションの対応もれ	通信機能	共有	調査過程において、BUS コマンドを使用しているモジュールを識別したら、モジュールの通信機能欄を『○』を記述する。	モジュール内の変更箇所と、通信機能との関わりをレビューする。変更箇所が通信機能を使用している事を認識し『●』にする。	『●』をチェックしたモジュールの変更点によってもたらされた通信機能の変化点が、『○』のモジュールに与えている影響が期待通りであることを確認するテスト設計を行う。 ・通信機能変更前後の振る舞いが互換であるか など	回帰テストの必要性を見出せることがポイントとなる	通信機能	見込める

表 A-4：問題事例への影響マトリクスの使用シミュレーション結果（4/4）

事例 No.	問題事例					影響マトリクスの使用手順 （『手順①カテゴリ・影響因子の検討』は各事例の共通作業として事前実施済み）					影響マトリクスの使用は、問題を検出するためのテスト設計への寄与が見込める？／見込めない？
	現象	変更点	原因	制約となるリソース	制約の分類	手順② 機能と影響因子の関連調査	手順③ 影響箇所の特定（変更要求 TM との対応付け）	手順④ 影響範囲の分析	影響範囲の分析のポイント	影響因子	
10	音声認識の一部のコマンドが認識されない。	ソフトバージョンアップ	ソフトバージョンアップ用の Disc を作成する過程で本来必要な File が抜けていた						ソフトバージョンアップで実現しようとした追加機能と抜け落ちてしまったファイルは、そもそも因果関係はなかった。Disc 作成時の問題である。 ソフトバージョンアップの機能差分の情報をもとに、影響範囲を探索しても、Disc 作成時の問題の検出は見込めないと考えられる。		見込めない