デジタルプラクティス　Vol.11 No.3(July 2020)

富士通ファミリ会論文

影響確認プロセスの定型化によるテスト品質向上の取組み

新井 雅之[1]　石田 保公[1]

[1]富士通エフ・アイ・ピー（株）

保守開発の現場では定期的にシステム改修が行われ，システム改修による妥当性を確保することに労力を費やしている．システム改修の妥当性は改修個所が正常に動作することだけではなく改修個所以外の影響を把握し，吸収して対応することも求められる．しかし実際の開発現場では周辺への影響確認まで手が回らず改修漏れとなるケースが多く見受けられる．本稿ではアプリケーション構造とテストシナリオの関係性に着目し，アプリケーション構造から機械的に改修の影響範囲を可視化したうえで，その範囲を網羅的にテストできる仕組みを構築した．また，一連の影響調査プロセスを作業フローとして定型化することで，現場に展開できる形として整理をした．保守開発案件に適用した結果，テストシナリオの不足による影響確認漏れの検出と影響確認のための必要十分なテストシナリオを導出し，テスト品質向上のプロセスとして確立できた．

※本稿は富士通ファミリ会2019年度優秀論文受賞論文です．

1. はじめに

1.1　当社の概要

富士通エフ・アイ・ピー（株）（以下FIP）では，数多くのお客様へのサービス提供で培った技術と，業種・業務ノウハウ，最先端技術を融合し，「システムインテグレーション」と「SaaS」の2つのサービスを核とし，「ITアウトソーシング&クラウドサービス」も組み合わせ，お客様に安心・安全で，高品質かつ高

コストパフォーマンスなディジタルサービスを提供している．専門性の高い業種・業務ノウハウを活かして，流通，製薬，ヘルスケア，官庁・自治体などのさまざまな分野の業種に対するパッケージソリューションやEDIサービスなどを展開している．

1.2　開発プロジェクトの特徴

当社のシステム開発プロジェクトの特徴として，新規開発よりも既存システムを運用しながらアプリケーションを改修していく保守開発の案件が多い．顧客に納入し運用しているシステムやパッケージに対して定期的に機能追加や要望取り込みを行うことに加えて，税法改正や緊急対応などで突発的に現行システムを改修する案件も発生している．そのため，FIPでも既存システムの消費税率変更への対応や元号改正への対応が求められ保守開発案件が直近3年間で3割程度増加している．本稿ではこうした保守開発の案件に焦点を当てて説明する．

保守開発では運用中の既存システムに影響を与えないことが求められる．しかし，アプリケーションの変更により変更個所以外の個所で以前は正常に動いていた機能に何らかの原因で障害が発生して品質が悪くなってしまう事象が発生してしまうことがある．本稿ではこれをデグレードと定義する．影響範囲の確認が不十分で本稼働後にデグレードが起こるイメージを**図1**に示す．デグレードは修正内容の影響範囲を把握したうえで必ずテスト段階で修正しなければならない．しかし，単純な機能追加のみの修正と認識していてもプログラムが構造化されていて参照や相関があるために，一見関係のないように見える機能でデグレードは発生する．そのため，修正内容の影響範囲を正確に把握しデグレードを防ぐことは昨今の構造化されたプログラムでは一筋縄ではいかないことが多い．

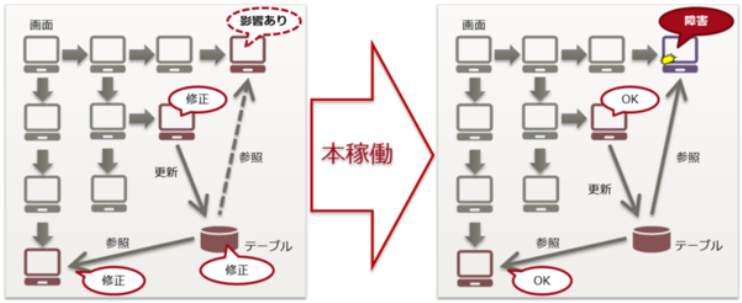

図1　影響範囲確認漏れによる障害のイメージ

以下，第2章では保守開発における問題点と課題について3つの観点に分けて説明する．第3章では影響範囲を確認し，テストするためのプロセスについて具体的に説明する．そして，第4章と第5章では本施策の評価と今後の課題についてまとめる．

2. 保守開発案件の問題点と課題

前章で述べたように保守開発においては，システム改修時の影響確認漏れによるデグレードが問題となっている．影響確認漏れが発生する要因となるプロジェクトの問題について，当社の直近3カ月の保守開発プロジェクトからヒアリングして分析をした．その結果，以下に示すとおり3つの観点で問題を分類することができた．

2.1 人の問題

アプリケーションの修正やテストをする担当者が影響範囲を判断できない，または誤ってしまうといった問題がある．理由として以下のようなことが挙げられた．

- 長年の保守開発でシステムが肥大化・複雑化しプログラムの中身を把握しきれない．
- プログラムとドキュメントの内容に乖離があり，ドキュメントが陳腐化していて正しい仕様を担当者が読み取れない．
- スキル不足や配置転換などにより担当者がシステムの仕様を理解できていない．

課題は誰が見ても影響範囲が明確に分かるようにすることである．人がアプリケーションの構造を把握できない状態になっているため，機械的にアプリケーション全体の関連性を可視化する必要がある．

2.2 時間の問題

影響範囲の確認または修正に対して，担当者の時間が確保できない点も問題として挙げられた．担当者の判断でテストする項目を端折ってしまった場合，結果としてテストしなかった部分がデグレードに繋がってしまう．時間が確保できない理由として以下のような点が挙げられた．

- テストすべき範囲が曖昧なため，網羅的に不要なテストをしてしまったり後から追加でテストが必要になるなど効率が悪い．
- 改修の都度，毎回すべてのテストを実施して動作確認することは時間を要

する.

課題は以下のとおりである．テスト対象を適切に選択することとテストの実施工数を削減することが目的である.

- 影響範囲が可視化され，どのテストをすれば網羅的に影響確認ができるか分かる.
- テスト自動化によりテスト実施時間を削減する.

特に保守開発案件では改修のたびに同じテストを繰り返し実行することになるため，テストコードによる自動テストの工数削減効果が高く有効である．ただし，テストコードの作成と保守にも工数がかかるため，繰り返しテストされる回数が多い業務上重要なシナリオをアプリケーション構造の観点から抽出できるようにして，その重要なテストシナリオを自動化するなどの工夫が必要である.

2.3　慣習の問題

アプリケーション修正時の影響調査や影響範囲のテストについては，有識者の勘や経験に頼る風習が根強いプロジェクトが多く，有識者に確認しないと対応を進められないといったこともよく見受けられる．本来は機械的に，網羅的に調査をしなければならないところを，有識者の見解のみを信用して終わらせてしまうような場合もある.

課題は影響調査やテストのプロセスを可視化し，それを標準化することで有識者に頼らないプロセスとして定着させることである．影響確認の属人性を排除することが重要である.

2.4　解決すべき課題

前節までで3つの観点でそれぞれの問題と課題について説明した．当社での現状を踏まえて解決すべき課題を以下にまとめる.

＜当社の現状＞

当社ではアプリケーション構造を把握するためのドキュメントとして，CRUD図や共通プログラム関連図を標準成果物として作成している．しかし，CRUD図や共通プログラム関連図などを実際の影響範囲調査にどのように利用するか，どのようにテストすれば影響範囲を漏れなく確認できるかといったプロセスが十分に整備されていないため，影響確認のプロセスにうまく活かし切れずに結局は有

識者を頼ってしまうことが多い．また，テスト工数を削減するためにテストコードによる自動化にも取り組んでいるが，こちらについても影響範囲が適切に把握できていないと繰り返し実行の恩恵を十分に受けることができない．

＜解決すべき課題＞

アプリケーション構造を把握するためのドキュメントやテストコード化などの取組みは行っているが，それを保守開発における修正内容の影響範囲を正確に把握しデグレードを防ぐための施策として落とし込めていないのが現状である．そのためアプリケーションの設計情報から資産修正の影響範囲を機械的に抽出し，どのテストを実行すれば影響範囲の確認が十分にできるかという作業を一連のプロセスとしてつなげて，それを誰でも確認できるようにすることを本取組みの課題として設定した．この課題に対応したのが次章で説明する「影響確認プロセスの定型化」の取組みである．

3. 影響確認プロセスの定型化

3.1 課題解決のアプローチ

他のアプリケーション資産に影響を与える資産として，アプリケーション構造上で相関関係の大きいデータベースとプログラム共通部品に焦点を当てた．焦点を当てた理由として複数の業務プログラムで同じデータベースのテーブルにアクセスする場合，そのテーブルの項目数や型が変わった場合には利用しているすべての業務プログラムが影響を受けるためである．プログラム共通部品についても同様のことが言える．修正が他の資産に影響を与えるイメージを**図2**に示す．

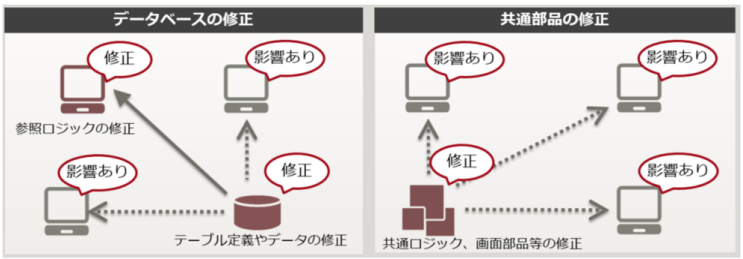

図2　修正が他に影響を与えるケース

アプリケーションの構造上重要なデータベースと共通部品と他のプログラムとの関連を把握する目的で，CRUD図や共通プログラム関連図を作成することは通常の開発でも行われており，SQLファイルやプログラムから自動生成するツールも一般化している．また，保守開発案件で影響範囲漏れを防ぎ品質を担保しながら開発を進めるXDDPと言った開発プロセスも提唱されている[1]．そのため，本稿でもCRUD図と共通プログラム関連図を使用してデータベースの修正，共通部品の修正に関する影響把握の方法として採用している．

本稿のアプローチのポイントは，アプリケーション構造の情報に対してテストシナリオを組み合わせる点である．可視化したアプリケーション構造から修正の影響範囲を把握し，影響を受ける機能を含むテストシナリオを抽出し，どのシナリオのテストを実施すれば影響個所を漏れなく修正できているかを確認できるようにした．影響確認プロセスのイメージを**図3**に示す．

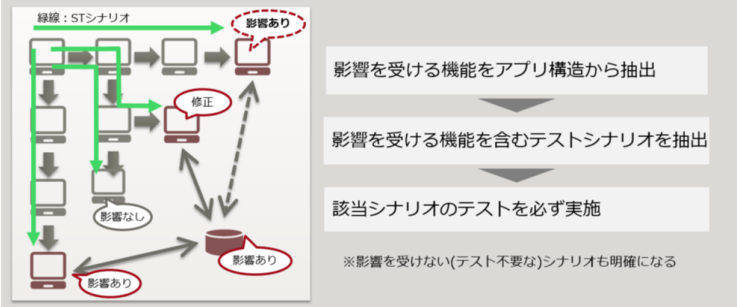

図3　テストシナリオを活用した影響確認

今回の取り組みの前提事項は以下のとおりである．

1．保守開発での適用
2．すべての機能の単体テスト（PT）レベルが完了していること

そのため，初期開発のSTで使用したシナリオを活用し，PTでは把握できない影響範囲をSTシナリオを通して検出することを目的とした．

3.2　全体概要

影響確認プロセスの定型化の全体像を**図4**に示す．全体を3つのステップに分けて説明する．

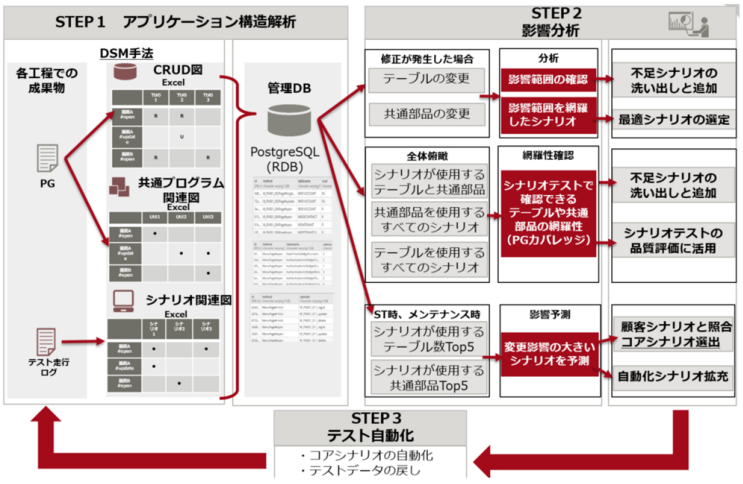

図4　影響確認プロセスの定型化　全体概要

3.3　アプリケーション構造解析

アプリケーション構造を可視化するための情報として，現行のプログラムからCRUD図と共通プログラム関連図を自動生成する．また，テストシナリオを打鍵したログからシナリオ関連図というドキュメントを自動生成する．これら3つのドキュメントはそれぞれ画面イベントの情報を持っており，画面イベントをキーにアプリケーション構造とテストシナリオの関係を可視化する．自動生成するツールが利用可能な前提条件として，現時点ではSpringベースのFIPのJavaフレームワークを使用しているシステムを対象としている．アプリケーション構造の解析と情報の蓄積までのイメージを**図5**に示し，それぞれについて説明する．

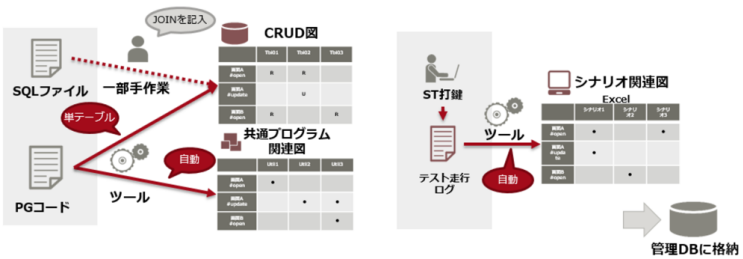

図5　アプリケーション構造の情報取得

(1) CRUD図の生成

まず初めに当社の既存ツールを利用して，Javaフレームワークの命名規

約に準じた画面とデータベースアクセス部品のクラス名とメソッド名から画面イベント単位のCRUDが分かるExcelファイルを生成する．単テーブルのアクセスについてはすべて自動でCRUD図を自動生成できる．結合しているテーブルの情報はクラス名から静的に取得できないため，SQLファイルの中身を見ながらテーブルの情報を手作業で追加する．結果として画面イベントごとにデータベースアクセスの数とCRUD種別が分かるようになるため，DB変更の影響を受けやすい画面イベントを可視化できる．

（2）共通プログラム関連図の生成

共通プログラム関連図は各画面イベントとプログラム共通部品の関係を表すExcelファイルであり，既存ツールを利用して完全に自動生成することができる．共通プログラムも決められた命名規約で作成しているため，CRUD図生成と同じ既存ツールを設定を変えて使用することで実現が可能である．自動生成された共通プログラム関連図から画面イベントごとに利用している（依存関係にある）共通プログラム名，呼び出し回数が分かるようになる．そのため，共通プログラムに変更があった際，依存関係があり影響を受けやすい画面イベントを可視化できる．

（3）シナリオ関連図の生成

シナリオ関連図はシナリオIDとそのシナリオが通る画面イベントの対応を表したExcelファイルである．テスト仕様書の項目からシナリオを抽出し，イベントと紐づけたシナリオ関連図を作成するのは工数がかかることが見込まれた．そのため，今回の取り組みの中でアプリケーションの打鍵ログからシナリオ関連図を自動生成するツールを作成した．自動生成の手順としてまず初めにテストシナリオに沿って画面を打鍵する．その結果実行された画面イベントがログとして記録されていく．次にその画面操作のログをインプットにして，ツールを用いてシナリオ関連図を自動生成する．テストシナリオの一連の操作と画面イベントが関連付けられるため，画面イベント単位でテストが網羅できているかどうかを可視化できる．

（4）解析情報の蓄積

前述の3つのExcelファイルをPostgreSQLの影響分析用データベース（以下，管理DB）に格納する．3つのExcelファイルを管理DBに格納する作業は，今回新規にツールを作成してExcelファイルをPostgreSQLのINSERT文に変換できるようにして効率化した．

3.4 影響分析

CRUD図，共通プログラム関連図，シナリオ関連図から抽出した情報を持つ管理DBに対して影響調査用のSQLを発行する．これら3つの情報は画面イベントをキーにして相互に関連付けされている．管理DBに格納されたデータのイメージを**図6**に示す．

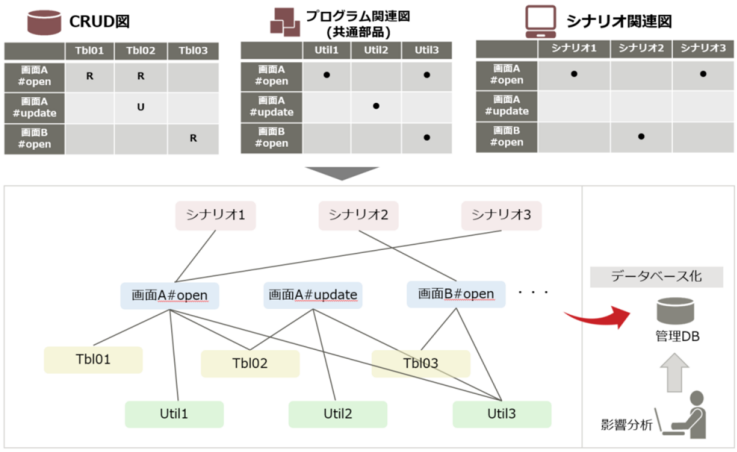

図6　影響分析のインプット（管理DB）

アプリケーション構造解析の結果を利用した影響分析について，実際の開発で考えられる2つのユースケースに分けて説明する．

ユースケース①

あるテーブルや共通部品に変更があった際に，影響を受けるテストシナリオを知りたい．

あるテーブルに変更があった際に影響を受ける画面イベントとテストシナリオを管理DBから検索した結果を**表1**に示す．データベースのあるテーブルに型変更などの修正を加えた場合，CRUD図の情報からそのテーブルにアクセスする画面とイベントが分かる．そしてシナリオ関連図の情報からその画面イベントをテストするテストシナリオまで辿ることができる．結果として表1のように画面イベントをキーにして，テーブルとテストシナリオが関連付けされる．

表1　テーブルを利用するシナリオ一覧

No.	テーブル	画面イベント	シナリオ
1	Tb101	画面 A#open	シナリオ１，シナリオ３
2	Tb102	画面 A#open	シナリオ１，シナリオ３
3	Tb102	画面 A#update	
4	Tb103	画面 B#open	シナリオ２

影響範囲として抽出されたテストシナリオを漏れなく実行することで，画面イベント単位での影響確認漏れを防ぐことができる．具体的には表1のNo.3のようにテーブルを利用する画面イベントはあるがテストシナリオが存在しない場合は，テストの実施が不足していることを示している．そういった場合にはテストシナリオの追加を検討する必要がある．

また，変更に影響しないテストシナリオも分かるため不要なテストを抑えることができる．表1のNo.1に示すように，Tbl01に変更があった場合にシナリオ2のテストをする必要がない．結果として影響範囲が分からないために盲目的に全量テストするといったことがなくなり効率的にテストができる．

ユースケース②

テーブルや共通部品の変更で影響を受けやすい（受けにくい）シナリオを探したい．

ユースケース①で示したように影響分析をすることでアプリケーション構造に基づく影響範囲が分かることに加えて，テーブルや共通部品を多く使用していて変更の影響を受けやすいテストシナリオと，反対にあまり利用していない影響を受けにくいシナリオを把握することができる．テーブルや共通部品の変更で影響を受けやすいシナリオについて管理DBを検索した結果の一例を**表2**に示す．昇順／降順ソートで件数順に整理することで，各シナリオでアクセスするテーブル，共通部品の名称と件数を確認できる．

表2　シナリオごとのテーブルアクセス，共通部品一覧

シナリオ	テーブル数	共通部品数	テーブル名	共通部品名
シナリオC	7	5	A, B, C, D, E, F, G	a, b, c, d, e
シナリオA	3	4	A, D, G	w, x, y, z
シナリオB	2	1	P, Q	A

また，分析結果をテスト自動化範囲の選定に活用するフローを**図7**に示す．今回の分析結果からプログラム観点で変更の影響を受けやすいシナリオ，受けにくいシナリオが特定可能となった．そこにお客様の日常的なオペレーションや業務観点で重要なシナリオを加えたものを「コアシナリオ」として定義し，このコアシナリオを初めに自動化する対象として選定する．すべてのシナリオをテストコードで自動化するのは，コードのメンテナンスも含めてコストがかかるが，影響分析の結果と合わせることで自動化する対象を戦略的に選定していくことができる．選定の方針については次節で説明する．

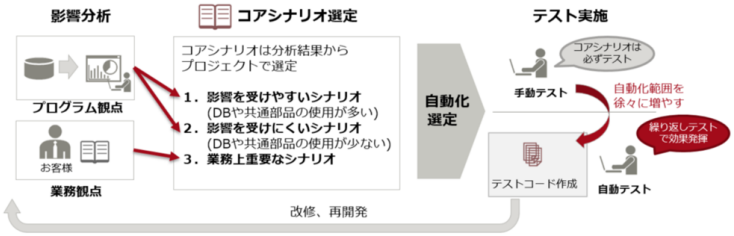

図7　影響分析のテスト活用フロー

3.5　テスト自動化

コアシナリオの画面操作を再現して自動でテストするために，Seleniumのテストコードを作成した．また，テストは繰り返し自動実行が可能となるように，データベースのデータを戻すスクリプトをコードの中に含めて冪等性を担保している．プログラム観点で自動化の対象を選定する際には下記の2つの考え方がある．

（1）デグレードが発生しやすい部分の品質を担保する

　　影響を受けやすいシナリオについてテストコードを作成する．修正のたびに影響を受ける範囲として上がり何度もテストしなければならないシナリオを自動化することで，確実にデグレードを検出する．その反面，テーブルや共通プログラムの変更に合わせてテストコードの修正も頻繁に発生する可能性が高くなる．

（2）想定していない範囲のデグレードを検出する

影響を受けやすいシナリオは人が必ずテストする方針として，人が気づかない（影響範囲だとは思わない）ような影響を受けにくいシナリオをテストコード化する．

どちらの方法を選択するにせよ，業務観点で重要なシナリオはデグレードが起こった場合の業務影響が大きい部分であり，毎回必ずテストをするべき対象として自動化を検討することが重要である．

4. 評価と考察

4.1 適用プロジェクト

今回の施策を適用したのは当社のEDIサービスの開発プロジェクトである．2次開発以降のスケジュールが決定しており，システム改修の際に既存システムの影響範囲を確認して自動テストができるように1次開発のST（システムテスト）行程で実現性検証をした．システム全体の規模は**表3**に示すとおりである．

表3　適用プロジェクトの情報

システム構成	Web(24 画面)＋API(12 本)
全体規模	31KStep
テーブル数	22 テーブル
共通部品数	57 本(関数単位)
シナリオ数	14 シナリオ(4 機能)

4.2 評価

本稿で解決すべき課題として設定した，アプリケーションの設計情報から資産修正の影響範囲を機械的に抽出し，どのテストを実行すれば影響範囲の確認が十分にできるかまでを把握できる一連のプロセスを確立することができたと考えている．その理由として，影響確認プロセスの定型化の3つのステップについて，それぞれの評価を次節以降で説明する．

4.2.1 アプリケーション構造解析の評価

アプリケーション構造を可視化するための情報として，CRUD図，共通部品関連図そしてシナリオ関連図を生成するフローを確立できた．シナリオ関連図の作成については，今回新規でアプリケーションログから画面イベントを抽出するツールを作成し，自動生成できる仕組みを構築した．システムの構造が複雑であったり，担当者のスキルが不足しているような場合でも，ツールを用いることで実際のプログラムから機械的にアプリケーション構造のデータを抽出できるようになった．

4.2.2 影響分析の評価

今回は影響分析を3.4節で説明した2つのユースケースに分けて検証した．

ユースケース①

あるテーブルや共通部品に変更があった際に，影響を受けるテストシナリオを知りたい．

検証の手順としてコアシナリオの1つで利用されている共通系のテーブル1つと業務系のテーブル1つ，計2つのテーブルに変更が入った場合を想定し，影響範囲をテストすることができるシナリオを管理DBから取得することにした．管理DBに検索SQLを発行した結果を**表4**に示す．テーブル列の対象のテーブルに対して，呼び出しのある画面イベントとその画面イベントをテストできるテストシナリオ，CRUD種別を順に示している．表4の内容から2つの結果を導き出すことができた．

表4　テーブル変更の影響を受けるシナリオ

No.	テーブル	画面イベント	シナリオ	CRUD
1	M004TENANT	W_FN001_004Page#open	ST1-001-001-001, ST1-002-001	R
2	M004TENANT	W_FN001_004Page#paging		R
3	M004TENANT	W_FN001_004Page#research	ST1-001-001-001, ST1-002-001	R
4	M004TENANT	W_FN001_004Page#search		R
5	M004TENANT	W_FN001_005Page#open	ST1-001-001-001, ST1-002-001	R
6	M004TENANT	W_FN001_005Page#regist	ST1-001-001-001, ST1-002-001	CRU
7	M004TENANT	W_FN001_006Page#open	ST1-001-001-001, ST1-001-001-003, ST1-002-001, ST1-002-003	R
8	T010NOTICE	P_FN014_001RestController #register	ST1-002-001, ST1-002-002, ST1-002-003, ST1-002-004	C
9	T010NOTICE	P_FN014_002RestController #register	ST1-002-001, ST1-002-002, ST1-002-003, ST1-002-004	C
10	T010NOTICE	W_FN002_014Page#open	ST1-001-001-001, ST1-002-001	R
11	T010NOTICE	W_FN002_015Page#open	ST1-001-001-001, ST1-001-001-002, ST1-001-001-003, ST1-001-001-004, ST1-002-001, ST1-002-002, ST1-002-003, ST1-002-004	R
12	T010NOTICE	W_FN002_016Page#open	ST1-001-001-001, ST1-001-001-002, ST1-001-001-003, ST1-001-001-004, ST1-002-001, ST1-002-002, ST1-002-003, ST1-002-004	R

（1）テスト不足の洗い出し

表4ではテーブルにアクセスしている画面イベントと対応したテストシナリオが記載されている．テストシナリオがない場合はテスト仕様書のパターン抽出が不十分で，テストの実施が漏れていることを示している．具体的には表4ではNo.2とNo.4が該当する．

（2）最適なテストシナリオの抽出

今回の2つのテーブルを利用するテストシナリオがシナリオ列に列挙されている．結果を見ると，ST1-002-001（下線あり）のシナリオを実行することでテーブルの変更影響を受けるすべてのイベントを網羅することが可能となる．なお，（1）で述べた不足しているテストは除いている．

結果として，不足しているテストの抽出とテーブル変更の影響を確認する際に必ず実施すべきシナリオの抽出（今回はST1-002-001というシナリオ1本）が検証できた．そのことから不足しているNo.2とNo.4の2つの画面イベント（W_FN001_004Page画面のページング処理と検索処理）をST1-002-001のシナリオの中で追加することで，1つのシナリオのテストでテーブル変更の影響確認が十分にできることが分かった．プログラム共通部品の変更についても，同様のユースケース検証により有用性を確認できた．

ユースケース②

テーブルや共通部品の変更で影響を受けやすい（受けにくい）シナリオを探したい．

検証の手順としてテーブルと共通部品の変更で影響を受けやすいシナリオを探し，コアシナリオを選定するケースを検証した．シナリオごとに利用しているテーブルと共通部品の数を表した一覧を**表5**に示す．今回はアクセスするテーブル，共通部品の多い上位8シナリオをコアシナリオに設定し，テストコード作成の対象とした．こうした結果から，アプリケーションの構造情報をもとに自動化するシナリオを選定するための情報を抽出することができた．

表5　シナリオで利用しているテーブルと共通部品

No.	シナリオ	テーブル数	共通部品数	テーブル名	共通部品名
1	ST1-002-001	15	52	M001ACCOUNT, M002VIEWSCOPE, …	CommonUtils#addErrorToNotice, CommonUtils#addRecordToReponse, …
2	ST1-001-001-003	15	43	以下略※	以下略※
3	ST1-001-001-004	15	42		
4	ST1-002-004	14	42		
5	ST1-001-001-001	14	40		
6	ST1-002-002	13	47		
7	ST1-001-001-002	13	37		
8	ST1-002-003	12	26		
9	ST1-011-002	3	13		
10	ST1-011-001	3	13		
11	ST1-012-001	3	12		

※各シナリオで利用するテーブル名と共通部品名を取得できるが，本稿では記載を省略する．

4.2.3　テスト自動化の評価

3.4節で影響分析をしたとおり，今回はコアシナリオ8本分のSeleniumのテストコードを作成した．テストコードについては，繰り返し実行してもデータの不整合が起きないように，データの戻しスクリプトを最初に実行するように組み込

んだ．作成したテストコードについては利用するテーブルと共通部品に変更があるたびに自動実行するようにしており，本稿を執筆した1次開発の時点では影響範囲にデグレードが起きていないことを自動で確認できている．

データ戻し処理も含めたテストコードの規模と，テスト対象のプログラム規模を**表6**に示す．今後ほかの保守開発案件でテストコード作成工数を見積る際の参考値として，元となるプログラム規模に対するテストコード規模の割合を算出した．テストコードの規模は，元となるプログラム規模の1.28倍という結果になった．テスト自動化の初回は元のプログラムと同等以上の規模のテストコードを作成する工数が必要になるが，自動化のメリットは繰り返しテストを実行していくことで表れるものである．繰り返しテストする際の自動化の工数削減メリットについては，テストコード作成や学習のコストを差し引いて4回以上繰り返すと効果が出ると一般的に言われている[2]．プログラム規模とテストコード規模を足し合わせたものをテスト規模と定義すると，テスト規模は元のプログラム規模の2.28倍になる．そのため，今回の場合は3回以上テストを繰り返した場合に工数削減の効果が表れることが期待でき，一般論とも大きな相違がないことを確認できた．回帰テストにおけるテストコード化のメリットは以前寄稿した論文も参考にされたい[3]．プロジェクトの2次開発はこれから開始する予定であり，自動テストを繰り返し実施することの工数削減効果とデグレードの検出については2次開発以降で検証する計画である．

表6　テストコードとプログラム規模

テストコード規模	9,076Step
プログラム規模	7,089Step
テストコード規模比率	1.28 倍

4.3　考察

第2章で保守開発案件の問題点を3つ取り上げた．それぞれの問題点に関して，今回の取組みの効果を考察する．

（1）人の問題

アプリケーションの中で他の資産に影響を与えやすいデータベースとプログラム共通部品に焦点を当てて，プログラムからCRUD図と共通プログラム関連図を生成した．プログラムの変更に合わせてこれらドキュメントを出し直すことで，プログラムとドキュメントとの乖離を防ぐことができ

る．そして，管理DBに登録したアプリケーション構造情報を検索することで，影響を受ける画面イベントを機械的に抽出することができるようになった．本稿ではさらにテストシナリオの情報と関連付けることで，どのテストを行えば影響範囲の確認ができるかというところまで踏み込んで可視化することができた．システムが複雑であったり，担当者のスキル不足などで影響範囲が分からないといった問題を十分に解決することができたと考えている．

（2）時間の問題

影響範囲の明確化により，影響が分からないために余計なテストをしてしまうといった問題を解決することができた．保守開発で繰り返しテストをする中でのテストコードの自動化による工数削減効果については，今後2次開発以降で検証していく．

また，影響調査プロセスの定型化の取組みを適用する場合の工数を見積もれるように参考までに，今回のプロジェクトで適用した際に集計した実績工数を**表7**に示す．プロセス確立までの試行錯誤や管理の工数を除外した実作業にかかる工数として，今回の規模で39.2人日必要になるという結果になった．そのほかには，初回のみの工数としてシナリオ関連図を生成するツールの作成に2人日かかった．保守開発を進める中で従来どおり繰り返し影響調査をする工数と本稿の施策を導入した場合の工数比較を今後実施していく．また，表7ではテストコード作成に工数が多くかかっているが，画面操作のスクリーンショットを取得して差分比較するなど改善の余地がある．

表7　本施策の適用工数

No.	作業項目	タスク	平均工数（人日）	本数	総工数（人日）
1	CRUD 図作成	CRUD 図作成	1.5	1	1.5
2	共通プログラム関連図作成	共通プログラム関連図作成	0.1	1	0.1
3	シナリオ関連図作成	走行ログの取得	0.4	25	10
4		シナリオ関連図作成	0.1	25	2.5
5	DB データ登録	DB データ登録	0.1	1	0.1
6	テストコード作成	機能別の初回作成	2.5	4	10
7		機能別の２本目以降	1.5	10	15
				計	39.2

（3）慣習の問題

管理DBにアクセスできる人であれば誰でも簡単な検索SQLを実行することで，影響範囲の確認をすることができる．そのため，プログラム観点の影響範囲については，有識者に確認せずとも任意のタイミングで確認することができるようになった．

5. 今後の課題

今回は実現性検証のフェーズでプロセスの確立するところまで進むことができた．今後は実際の保守案件プロジェクトで適用して実績を蓄積していきたい．また，技術的な課題として以下の3点を掲げ，プロジェクト適用と並行して取り組んでいく．

（1）データ利活用

CRUD図や共通プログラム関連図などの静的な関連情報のほかにも，以下のような運用中のデータを活用して，コアシナリオの選定や効果的なテストの実現に繋げていきたい．

・構成管理情報から，頻繁に変更される資産を可視化
・アプリケーションログから頻繁に利用されるシナリオを可視化
・障害管理情報と連携し，品質の可視化

（2）ユーザビリティ向上

影響範囲を確認するために，現状は管理DBに対して検索SQLを実行する必要がある．これに対して，誰でも検索できるようにツール化やブラウザから影響範囲を確認できるようにする．関連性をグラフィカルに可視化するために，グラフDB（Neo4j）の利用を検討している．**図8**に画面に表示されるサンプルを示す．ある画面イベントのオブジェクトを選択すると，利用するデータベースのテーブルや共通部品，テストシナリオがツリーで表示され，関連を辿っていくことができる．

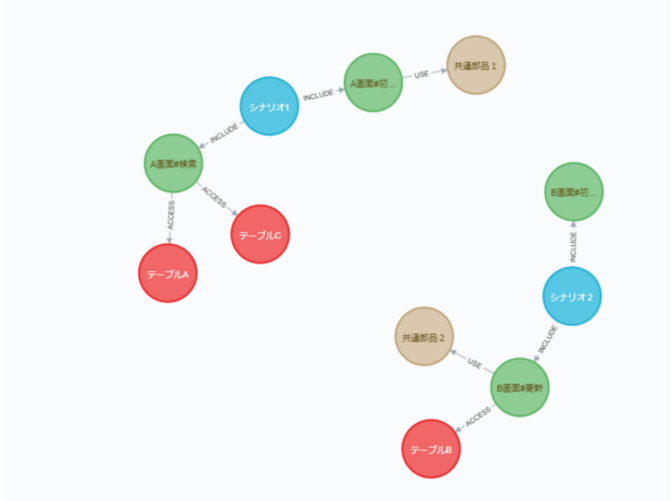

図8　グラフDBを利用した関連性の可視化

（3）AIの活用

テスト自動化の課題として，テストコードの保守コストがかかることが挙げられる．富士通の「AI for Testing」の取組み[4]と連携して，AIを活用したテストコードの保守効率化を目指す．「AI for Testing」では，以下の2点の実現を目指している．

・アプリケーションの変更を検知してテストコード自動修復（セルフヒーリング）
・AIによるテスト自動生成と実行（セルフテスティング）

6. おわりに

当社のビジネスを支えている保守開発プロジェクトに向けて，業種によらず課題となっているシステム改修時の影響確認漏れを解決するアプローチを形にすることができた．CRUD図などを活用してアプリケーションの関連性を可視化する取組みは各方面で行われているが，今回の取組みではそれをテストシナリオと掛け合わせることで，影響範囲を漏れなくテストするという具体的なアクションまで繋げられるところが訴求点である．また，アプリケーション構造の情報抽出について大部分をツールを用いて自動化したことで，現場プロジェクトで導入する際のコストを低減することができたと考えている．本稿の影響確認プロセスを標準的な手法として，そこに業種やプロジェクトごとの観点を付加し，全社的な影

響調査の手法として展開し，品質向上に貢献していきたいと考えている．まずは社内の保守開発をしているプロジェクトに浸透させ，今後はグループ会社や社外にも展開していきたい．

謝辞　影響確認プロセスの定型化の取組みにあたり，実現方式の検討段階から多くの知識や示唆をいただきました原部長に深謝いたします．そして本稿を作成するにあたり，ご指導をいただいた標準化推進部の諸先輩方に感謝いたします．

また，本取組みについてご理解をいただき，実現性検証の実施を快諾して下さったEDIビジネス部の宮崎部長，村田シニアマネージャに心から感謝の気持ちと御礼を申し上げます．

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，Chapter2 (Nov. 2007).
2）板垣真太郎：AI for Testingのスタートアップ情報共有，テスト自動化カンファレンス2018，富士通（株），p.30 (Jun. 2018)
https://www.slideshare.net/ssuser5c492c/ai-for-software-testing
3）新井雅之：新旧比較照合テスト自動化によるマイグレーション開発の品質保証，第28回プロジェクトマネジメント学会，富士通エフ・アイ・ピー（株），p.6 (sep. 2016).
4）板垣真太郎：前掲　AI for Testing のスタートアップ情報共有，pp.10-29.

新井 雅之（非会員）arai.masayuk-01@jp.fujitsu.com
　2009年 富士通エフ・アイ・ピー（株）入社．2019年 現在 標準化推進部所属．

石田 保公（非会員）yas.ishida@fujitsu.com
　2002年 富士通エフ・アイ・ピー（株）入社．2019年 現在 標準化推進部 マネージャー．

採録決定：2020年2月26日
編集担当：斎藤　彰宏（日本アイ・ビー・エム（株））