大規模開発における生産性向上策実践事例の紹介

Introduction of Practical Examples of Productivity Improvement Measures in Large-scale Development

寺尾和樹，滝之入芳輝

要　約　システム開発では様々な生産性向上策が存在し，大規模開発への適用事例はコストへの影響が大きいためナレッジ化しておくことが有効である．本稿では大規模開発のプロジェクト実践事例として「プログラムレス開発ツールであるODIPのBankVision®への適用事例」と「リスクベースドテストの実践事例」について紹介する．実践事例を通じて，生産性向上とコスト最適化の観点から適用効果があることが確認できた．また，今後のプロジェクトでの適用効果を上げるための考察を行った．

Abstract　There are various productivity improvement measures in system development, and it is effective to make them explicit knowledge because the application cases for large-scale development have a large impact on costs. This paper introduces “application examples of ODIP, which is a programless development tool to BankVision®” and “practical examples of risk-based testing” as practical examples of large-scale development projects. Through these examples, it was confirmed that there was an application effect from the viewpoint of productivity improvement and cost optimization. Also, consideration was given to improve the application effect in future projects.

1. はじめに

社会インフラとなった情報システムにおいて，システム化ニーズの増加及び労働人口減少の背景もあり，生産性向上への社会的要請は留まることがない．それを受け，システム開発では様々な生産性向上策が存在する．しかし大規模開発への適用事例がナレッジ（形式知）化されている例は少なく，適切な生産性向上策の選択が難しくなっており，それは大規模開発の事例が多い金融分野でも同様である．

本稿ではBIPROGY株式会社（ビプロジー株式会社．以降，BIPROGY）が生産性向上にむけたナレッジ化を目的として，金融大規模プロジェクトにて実践した二つの生産性向上施策とその実践事例について紹介する．一つ目は，プログラムレス開発ツールの導入を行い，作業効率を向上させる施策である．2章にて，開発の実装量を減らすことができるプログラムレス開発ツールであるODIP[*1][1]のBankVision®[*2]への適用結果について述べる．二つ目は，テストによって排除できるリスクを可視化し，それを基にテスト範囲を調整することで，無駄なテストケースを省く施策である．3章にて，リスクとコストの観点からテスト優先度を決定するテスト手法であるリスクベースドテストの適用結果について述べる．

2. 事例概要～プログラムレス開発ツールによるコーディング負荷軽減～

BankVisionのバッチ処理開発では，オープンプラットフォームの技術とツールを活用し，

データベースアクセスや帳票出力といった部分でコーディングを簡素化する手法を確立している．その上で，オープン技術・ツールを有効活用して更なる生産性向上を図る取り組みを継続的に行っている．ノーコード・ローコード開発（プログラムレス開発）を実現するツールが普及する中，BankVision 導入時の大規模開発において，プログラムレス開発ツール ODIP を利用してバッチ処理開発を実施した．本章では，このプログラムレス開発における生産性向上事例について述べる．

2. 1　プログラムレス開発ツール ODIP の概要

ODIP は Java ベースの高速開発ツールであり，以下の三つの特長がある．

1）モデルドリブン・アーキテクチャ[*3] での処理実装ができるためプログラミングが不要

2）GUI 画面で開発することができる

3）テーブル・項目の一元管理により項目の追加・廃止や属性変更が簡易

このような特長により，ODIP は BankVision の現行開発言語である COBOL に比べてコーディング負荷が小さく，開発作業の生産性向上に寄与する．

2. 2　BankVision へ ODIP を適用する際の工夫

勘定系システムへの ODIP の適用にあたり，大規模開発での活用効果を高める工夫として，以下 1）～ 4）の対応を行った．

1）既存資産の有効活用

ODIP で開発したアプリケーションから，BankVision の既存資産である利息計算サブルーチンなどの共通処理を呼び出すために，既存開発資産である「業務サブルーチン（COBOL）」と「MIDMOST[*4] DB サービス」を呼び出すためのプログラム連携インターフェースを開発した．

2）トランザクションの一貫性の確保

BankVision の COBOL アプリケーションは，オープンミドルウェア「MIDMOST」経由でデータベースにアクセスすることで同一バッチプロセス単位のトランザクションの一貫性を確保している．既存の ODIP では，MIDMOST と異なり JDBC により DB 接続が実装されており，SQL 実行の都度，SQL 実行用のコネクションを取得し，その SQL の終了でコネクションを解放するとともにコミットを発行している．つまり，同一バッチプロセス内でトランザクションの一貫性が保証されていない実装となっている．ODIP にデータソースとして MIDMOST を追加し，MIDMOST の DB サービスを活用してコネクションを引き継ぎながら DB にアクセスするように ODIP を改善し，トランザクションの一貫性を確保した．

3）設計パターンの制定

ODIP での開発に適しているバッチ処理の設計パターンを整理し，資料として制定した．その資料を設計要員に展開することで，ODIP での実装が困難な処理や COBOL での実装に比べて生産性が低下する機能には適用しないようにする対策を行った．

4）教材・カリキュラムの整備

ODIP に関する知識がない開発要員向けに，教材と育成カリキュラムの整備を行った．解析と修正の実践スキルを 8 時間で習得する講習カリキュラムを整備した．

2. 3　プログラムレス開発ツールの開発生産性の評価

ODIP を適用した大規模開発プロジェクトの事例では，バッチ処理 712 本を開発した．この開発実績を基に，ODIP 開発の生産性基準を定めている．この生産性基準を，BankVision での COBOL 開発の生産性基準と比較することで，ODIP 適用による生産性向上の評価を行った．

2. 3. 1　評価方法

BankVision の COBOL 開発の生産性基準においては，処理の開発「難易度」をランク付けしている．COBOL 開発の基準に対して，ODIP 開発における難易度の基準を対比させることで，同等条件の難易度における開発生産性（詳細設計，製造，単体テスト工数）を比較した．

COBOL 開発の生産性基準における難易度の基準に対しては目安として COBOL の STEP 数が示されており，この STEP 数に開発工数が依存するという考え方である．一方，ODIP における処理単位は「管理単位」と呼ばれる処理設定の定義体である．ODIP はプログラムレスであるため STEP 数という規模指標は存在しないが，開発工数は管理単位数に依存するという考え方を取ることができる．管理単位は複雑な出力処理を行う場合に，COBOL 開発での難易度の上昇に対して，管理単位数の上昇幅が大きくなる傾向がある．

COBOL 開発の開発生産性基準における難易度に対して，ODIP 開発の場合に用いる管理単位数を目安として対比したものが表 1 である．

表 1　COBOL の開発難易度と ODIP の管理単位数の対比

BankVision の COBOL バッチ開発生産性基準			ODIP の管理単位数
難易度	難易度の基準	COBOL STEP 数	
E	計算や複雑な編集を伴わない非常に小さい処理など	2000 未満相当	1 ～ 2
D	入力ファイルが単数で，計算や複雑な編集を伴わない単純なデータ抽出処理など		3 ～ 4
C	入力ファイルが 3 種類未満のバッチ処理，通常の帳票作成処理など	2000 以上相当	5 ～ 6
B	入力ファイルが 3 種類以上のバッチ処理，または複雑な帳票（明細定義が複数ある）の作成処理など	2500 以上相当	7 ～ 10
A	計算ロジックのある顧客宛て帳票作成処理，または入力ファイルが 5 種類以上のバッチ処理など	3000 以上相当	11 ～

2. 3. 2　ODIP 開発生産性の構成

BankVision 開発における ODIP の開発生産性は，作業ごとに表 2 の構成となった[*5]．この開発生産性は，基本設計において開発が必要と導出した論理的な処理（おもに最終出力[*6]）を算出単位としている．この処理単位に対して，詳細設計で処理単位が管理単位に細分化される

が，表2において「管理単位作業(※)」列に「○」が付されている作業は，生産性の値に分割された管理単位数を乗じて生産性を求める構成となっている（例えば管理単位数が「2」のODIPを開発する場合，詳細設計工程の「中間テーブル設計」作業は4（2×2）要するということを表している）．

表2　ODIP開発生産性基準（単位：プログラム仕様書（ODIP版）作成にかかる時間を1とする）

工　程	作業	作　業　内　容	管理単位作業(※)	生産性
詳細設計工程	作成	プログラム仕様書（ODIP版）作成	—	1
		管理単位設計	—	5
		中間テーブル設計	○	2
		ジョブネットフロー図作成	—	1
	検証	詳細設計工程の作成作業の総工数の20%とする．		
製造工程	作成	管理単位定義作成	○	1
		入力定義作成	○	1
		加工定義作成	○	2
		出力定義作成	○	1
	検証	製造工程の検証は単体テストの検証と同時に行う．		
単体テスト	作成	単体テストケース／テスト用データ作成	—	3
		単体テスト実行	○	3
		単体テスト結果確認	○	2
	検証	単体テストの作成作業の総工数の20%とする．		

2.3.3　開発生産性の評価結果

表2のODIP開発生産性基準を基に管理単位数毎のODIPの開発工数を算出し，表1のCOBOLの開発難易度とODIPの管理単位数の対比表に基づき，同一難易度におけるODIPの「開発工数」とBankVisionの「COBOL開発生産性基準」を比較した結果が表3である．それぞれの生産性については非公表のため，定性的な結果での記載としている．難易度「A」の管理単位数については上限値を定めていないため，生産性向上効果が発生する管理単位数まで表記している．

ODIPの適用対象としては管理単位数が11以下となるバッチ処理の場合が，開発生産性向上の目的に対して有効であることが分かった．2.3.1項で述べた通り，複雑な出力のバッチ処理の場合は管理単位数の上昇幅が大きくなるため，単純な出力のバッチ処理への適用が適している．

開発工程以外の生産性については，開発言語による作業内容に相違がないため，開発生産性の差異は発生しない．

表 3　開発生産性基準比較

ODIP 管理単位数	COBOL 難易度	生産性向上効果
1 ～ 2	E	↑（生産性向上）
3 ～ 4	D	↑（生産性向上）
5 ～ 6	C	↑（生産性向上）
7 ～ 10	B	↑（生産性向上）
11	A	↑（生産性向上）
12		—（差異なし）
13 ～		↓（生産性低下）

2. 4　さらなる生産性向上に向けた考察

本章でみてきたように，BankVision のバッチ処理開発に対するプログラムレス開発ツール導入は，一定の開発生産性向上効果があることを確認できた．その上で，プログラムレス開発ツールの特性を活かしてさらに生産性を向上させる方策を考察した．以下 1）と 2）に挙げる．

1）プログラムレス開発ツールの効果を高めるデータ環境のあり方について

BankVision のバッチ処理開発における入力 DB 群は「共用明細 DB[*7]」を基本としている．共用明細 DB は，オンライン DB の素データに近い内容なので，業務的に意味のある最終出力項目の編集処理が複雑になるケースが多くなる．

そのため，2. 3. 2 項で述べた BankVision における ODIP 開発の事例から導き出された開発生産性基準は，共用明細 DB に対して一定のデータ加工を行った後に最終出力項目を編集することを前提としている．入力 DB 群に最終出力項目として使用されるデータ（業務的に意味のあるデータ）を充実させ，各処理でデータ加工作業を削減することでプログラムレス開発の生産性向上が見込める．

2）入力 DB 群を拡充させる方法

業務的に意味のある入力 DB とするための具体策としては，共用明細 DB 群に，業務的に有意義な DB 項目を派生属性として追加し，業務的説明を加えたデータ辞書，項目編集要領を定義し公開するという対応が考えられる．業務的に有意義な DB 項目にふさわしい情報とは，例えばユーザが参照する帳票出力項目など業務に直接使用される情報が挙げられる．継続的な入力 DB 群の拡充を行っていくことで，導入後も生産性の向上が見込める．

3.　大規模開発のリスクベースドテストの実践事例

リスクベースドテスト設計とは，リスク対処にかかるコストを基にテスト優先度を決めることで，コストを最適化する手法である．本章では勘定系システムの更改案件を事例とし，リスクベースドテストの実践に向けた作業を説明する．

3.1　事例概要

当該案件は，勘定系システムのハードウェア/ミドルウェアのサポート停止に伴いハードウェア機器の更改，ミドルウェアのバージョンアップ対応を実施したものである．本案件では，ミドルウェアバージョンアップに伴う非互換リスクの調査に時間を割かずに開発コストを下げコスト算出までのスピードを重視する顧客と，非互換リスクが見えていないため明確なコスト算出ができないBIPROGYとの間で案件着手が進まない状態であった．非互換リスク調査の結果によりコスト算出に大きな影響が発生するリスクを顧客側に継続的に説明することで，非互換によるリスク明確化とコスト算出を目的とした構想フェーズを立ち上げ，顧客とリスクベースドテスト手法に基づき，適正コストによるテスト方針を検討した．

3.2　リスクベースドテストとは

リスクベースドテストは，テスト作業の優先順位を決めることでプロダクトリスクのレベルを低減させ，ステークホルダにその状態を通知するテストの方法である．ソフトウェア品質を向上するためのテストは不可欠であると同時に，多くのコストと時間を要する．しかしながら，開発スケジュールやコスト等には制約があり，ソフトウェアテストを際限なく継続することはできないため，テスト作業を効率化し，制約の中でテスト品質を確保することになる．このような場合，リスクベースドテストは現実的な対応策である．リスクベースドテストではプロダクトリスクの重大度と発生率を整理することでリスクを可視化し，システム品質を保ちながらテスト削減を行うことでテスト効率を向上する．

3.3　リスクベースドテスト設計に至る作業プロセス

本節では，リスクベースドテスト設計の作業プロセスを表4に整理した上で，勘定系システムの更改案件のテスト設計を実践事例とした作業プロセスや考慮点を各項で説明する．なお，リスクベースドテストを用いた設計作業が後工程になるほど，リスクが残存する可能性が高くなるため，より上流工程で実践することで適用効果が高くなる．

表4　リスクベースドテスト設計の作業プロセス概要

作業プロセス	考　慮　点
案件特性の把握	1）作業前提/懸念事項の整理
リスクマトリックスの作成	1）作業についてのリスク項目の洗い出し 2）リスク評価基準の設定 3）リスク項目ごとに基準を用いてリスクを評価 　リスクに対しての対策要否の検討
テスト方針の確定	1）テストステークホルダの整理 2）リスクマトリックスの評価 3）テスト範囲の確定

3.3.1　案件特性の把握

案件の要件に対してどのようなシステム対応が適しているのか，その対応にどのような問題が埋め込まれる恐れがあるのか，案件関係者はシステム開発作業が始まる前にこれらの要素について把握をしておくべきである．案件を進めていく中で，ゴールの見えない作業に対して見

積りを進めることは即ち，見積り内容に大きなリスクがある状態でプロジェクトを推進するということである．そのため，テスト設計を始める前には「案件対応の前提事項」と「テスト作業を見積りする際の懸念事項」について明確化することが重要である．

1）作業前提/懸念事項の整理

顧客の要望や要件に対して，前提事項が明示的に記載されているものもあれば，暗黙的な前提もある．プロジェクトには複数の関係者が存在し，それぞれの立場によって異なる目的や思想や経験を持っているので，暗黙的な前提は認識の不一致を内在している恐れがある．そのため，案件を進める上では前提が崩れた場合に備えて「コスト」に影響のある前提の洗い出し作業を実施しておく．また有識者にヒアリングを行って表面化していない懸念事項についても整理し，検討要否の期限を明確化する．作業前提は共有されていないことが多いため，本作業はプロジェクト関係者で共有し，漏れなくヒアリングをすることが重要である．

3.3.2　リスクマトリックスの作成

リスクマトリックスとは，テスト対象の機能に対し「リスク発生の可能性」と「リスク発生時の影響度」からリスクを定義し，機能毎に評価を行うための表のことである．本項ではリスク定義を行う上での要点を説明する．

1）リスク項目の洗い出し

リスクは，顧客，ユーザ，テスト関係者等のステークホルダのプロダクト品質またはプロジェクト成功に対する認識に悪影響を与えるものである．前項で述べたようにステークホルダによって前提の置き方が異なるため，ステークホルダ自身がリスクを列挙するのではなく，テスト設計者が質問形式で関係者に確認し，客観的にリスク項目を洗い出すことが重要である．リスクを洗い出す際には「システム対応による変更点」と「想定される具体的な問題事例」を組み合わせて，リスクを一覧化する．

2）リスク評価基準の設定

テスト対象の機能に対し「ⅰ）リスク発生の可能性」と「ⅱ）リスク発生時の影響度」の観点から評価基準を定義する．

ⅰ）リスク発生の可能性

システム特性（事前の机上調査やOS/ミドルウェアのプロダクトセット選定）や顧客の特性を踏まえて「未来に発生するかもしれない起こってほしくない事象」が発生する可能性を，表5の評価分類例に沿って整理する．リスクの「発生する可能性」（発生頻度）が予測できるケースの方が少なく，発生可能性を定量的に評価することは困難であるため，「机上調査によるリスクを否定する情報の有無」「過去事例の有無」から評価する．

表 5　リスク発生可能性の評価分類例

評価	評価項目の説明
大	リスク発生を否定する材料が皆無であり，発生する可能性が極めて高いもの．または影響判断できないもの
中	事前調査結果の情報が曖昧であったり，不明瞭であるため，リスク発生の可能性が否定できないもの．また過去同様の対応事例がないもの
小	事前調査の結果リスクが明確に否定されているが，事例として実績がなく検証を要するもの
無	事例での実績や経験があり，一般的な開発プロセス（有識者による設計～実装）でリスク回避できるもの

ⅱ）リスク発生時の影響度

開発工程で問題が起きた場合に，「修正範囲」「修正の規模（工数）」の程度を，表 6 の評価分類例を基に評価する．並びに本番稼働後に当該事象が見つかった場合のステークホルダへの影響について，表 7 の例に沿って評価する．本番稼働後の影響範囲については，上記観点に加え「復旧の可否」「障害が発生した場合の頻度」からも評価する．

表 6　開発中の影響範囲評価分類例

評価	評価項目の説明
大	リスク発生時の対応がシステム全体に波及するもの．あるいは，手戻りや横展開作業が 15 人月を超過する見込みであるもの
中	リスク発生時の対応がミドルウェア設計やアーキテクト等に波及するもの．あるいは，手戻りや横展開作業が 10 人月を超過する見込みであるもの
小	単体機能，プログラム内に収まるもの．あるいは，手戻りや横展開作業が 3 人月を超過する見込みであるもの
無	運用設計等で考慮するもの．あるいは，手戻りや横展開作業が 3 人月に満たないもの

表 7　本番稼働後の影響範囲評価分類例

評価	評価項目の説明
大	システムダウンやシステム全体のスローダウン，復旧困難なデータ破壊を伴う障害
中	個別トランザクション，バッチのエラーや部分的なデータ欠損を伴う障害
小	低頻度で発生するエラーや業務影響なしで運用回避できる障害
無	警告メッセージの追加等，業務運用上変更に影響を受けないもの

3）リスク評価/対策要否の検討

リスク項目に対し，リスク発生の可能性と影響範囲の観点からリスク評価を実施する．リスク評価の観点として，開発プロセスの中で対策をしない場合に，どの工程で検出できる問題なのか確認する．また，開発工程着手前に確認できる内容と，対応した場合のコスト規模感についても整理する．リスク評価を基にリスク重大度や対策の有無を共有することで，個別に追加対応すべきリスクなのかを協議する．リスク対策については，追加テストをする以外に，実施しないという判断が取れる．その分，リスク重要度が高いものに対しては，十分

なカバレッジが網羅できるようになり，テストアプローチを選択する柔軟なテスト戦略に繋がる．重要なのはシステム構成を理解し，事前に把握しているリスクについて対応方針を明確化し，リスク対処にかかるコストの必要性を関係者に把握させることである．また対応しないと判断した場合は，残存するリスクについて顧客の意思決定者に理解してもらうことである．テスト設計者はリスクをゼロにするのではなく，検知したリスクとその対策内容について顧客を主としたステークホルダの理解を得るための説明責任が求められる．

3.3.3　テスト方針の決定

リスクの大きさに応じて，対応優先順位，コスト，要員リソース等の観点からテスト実施スコープを確定する．リスク評価で検討したリスク対策に対してコスト捻出の妥当性が理解されれば，リスク対策に要するコストを組み込んだ作業計画を立てることができる．一番大きなリスクを持つ欠陥から順に検出できるようにテストを決める．最終的に残置されたリスクについては顧客責任者が対策の可否をリスク重要度と捻出可能コストから決定する．

1）テストステークホルダの整理

テスト設計を行う際にテストの役割を定義し書面化することは，後工程での問題を防ぐ上でも非常に重要である．役割が明確化されていないことで意思決定の範囲が不明瞭になり，方針が決まらなくなることが問題となる．重要なのはテストを実施/推進する上で役割と責任範囲を決めるための前提を定義することである．顧客文化によりこの責任範囲の考え方は大きく異なる．また顧客内であっても，個人の思想により期待値が異なり顧客の想定と相違する危険性がある．この定義が異なることで責任範囲のずれによる予想外のトラブルが発生する懸念があり，余計な作業負荷や責任が発生する恐れがある．テストのステークホルダは，プロジェクト，プロダクト，組織，およびその他の要因によってさまざまである．表8に示す役割を定義し，作業分担を実施した．

表8　役割の定義について

役　割	担　当　者	役割の説明
実行責任	作業担当者，チームリーダ 顧客担当者	テスト実行フェーズにおける実担当者．並びに検証責任者
説明責任	テストマネージャ	テスト計画作成者．テスト作業を説明し，テスト作業の必要性や品質確保の方法について説明を行う
相談者	プロジェクトマネージャ 上位責任者（顧客，ベンダ） チームリーダ 作業担当者	テスト方法やスケジュール，要員調整等の相談先．基本的にテストステークホルダ
意思決定者	顧客上位責任者 プロジェクトマネージャ ベンダ上位責任者	担当者からの説明を確認し，その方針に対して意思決定を行う

2）リスクマトリックスの評価

ステークホルダとリスクマトリックス（図1）の評価を行い，リスク項目と評価指標につ

いて共有する．本作業を行う上での確認観点は 3.3.2 項 3)「リスク評価/対策要否の検討」と同様の作業項目となる．本作業を実施する中での違いは，想定しうる問題が発生した場合の対応コストについて顧客認識を把握し，案件コストの最大値を共有することである．リスク対策を行う上で，「開発プロセスの改善」や「開発者の適切なトレーニング」といった，テスト以外の活動でリスクを減らす方法があるか検討することが求められる．

プログラム起点で発生する不具合の発生原因の分類	影響範囲				リスク対策要否	必要テスト対象を選定 凡例：●：全量，○：非網羅，－：非対象						
	影響範囲	評価理由	発生可能性	評価理由		製造	単体テスト	結合テスト	機能テスト	障害	運用	性能
I.非互換（バージョンアップ）によるもの												
コンパイル時のエラー												
コンパイルチェックの厳密化により、非推奨ロジックの残置によりコンパイルエラーが発生	大（X人月）	コンパイル時のエラーは構文エラー（シンタックスエラー）であることが想定。発現時はプログラム全体へ影響する可能性がある。	中	机上確認の結果互換性に関する言及があり、可能性は低いものの当社実績は未確認。	リスク発動時のコストが大であり規模点での見積もりが困難であるため、実機確認でリスク低減が必要。リスク対策として非互換確認の一環でVerのCOBOLを開発環境に導入し全プログラムのオールリコンパイル確認を実施する事でリスク軽減を行う。	●（UAT）	－	－	－	－	－	－
II.M-Wやアーキテクト間の非互換（OS（Windows）・M/W（MQ、HULFT、JP1、SQLServer））												
機能（COBOLプログラム）からのM/W呼び出し失敗												
非互換影響を受け、プログラムからM/W呼び出し（MQ送信、HULFT送信、JP1実行等）の処理に失敗する。	小（X人月）	ミドルウェア間の接続は、共通部品に集約しているため、発生時の影響は限定的。	大	ミドルウェア間の親和性評価（プロダクトセット選定）は未完了であり、転方のプランはあるものの共通部品への影響は発生する可能性あり。	リスク発動時の影響範囲は小さいもの、机上確認にてミドルウェアの調査を行う事で防止できるリスクとなる為、非互換調査を実施する事でリスク低減を行う。またテスト工程においては共通部品のテストを重点的に確認する。	－	－	●（UAT）	●（UAT）	●（UAT）	●（UAT）	●（UAT）

図 1　リスクマトリックス（抜粋）

3）テスト範囲の確定

リスクの大きさに応じて，優先順位，コスト，要員リソース，時間等の観点からテスト実施スコープを確定する．類似案件の実績を基にシステム構築を実施した場合に使用するテスト項目を一覧化した上で，どのような要素（プログラム，機能，システム，手順書）の品質保証を目的にしたテストか，誰がそのテストに関係するのかを整理する．また，一番大きなリスクを持つ欠陥から順に検出できるようにテストスケジュールを決めた上で，最終的に残置されたリスクについては顧客責任者が対策の可否をリスク重要度と捻出可能コストから決定する．重要なのは，テストの必要性や効率的なアプローチについてテストマネージャは説明を実施するに留まり，テスト実施可否は顧客意思決定者が判断するということである．テストを実施しないと判断をした場合には，少なからず品質が落ちる恐れがあるため，その品質確保にコストをかけるかは顧客側の意思決定者の判断に委ねられる．なお当該テストを実施しないと判断するには，リスク評価で実施した，どの工程でその問題が検知できるのか．問題が発生した場合にどのような影響があるのかの説明が欠かせない．

3.4　リスクベースドテスト適用評価

当該案件ではテスト対象の機能要件定義前にリスクベースドテストに基づいたテスト計画を実行する中で，結果として不要なテストを削減しながら優先すべきテストを重点的に行うことで，品質を確保しながらも，当該案件の開発コストを顧客要望と一致させることができた．テスト計画の成功要因としては顧客が BIPROGY の考える非互換リスクについて理解を示し，非互換調査への着手を承認したことが挙げられる．非互換調査前には業務アプリケーションの非互換影響が見えておらず，アプリケーションテストの優先度が高いとの顧客認識があったことから全機能の網羅テストを要件に組み込んでいたが，過去の同様な事例の実績と机上調査からアプリケーション修正が不要な可能性が高いことが分かった．また，当該システムではミドルウェアとアプリケーションの処理パターンを網羅的に整理した共通フレームワークを定義し

ており，該当機能のテストの網羅性を担保することで業務アプリケーションの非互換リスクを極小化できることが分かったため，フレームワークテストの優先度とテスト密度を上げ，業務アプリケーションについては起動確認のみの最小限でのテストとする方針とした．

3.5　今後に向けた提言

システムには，開発時の部分的なバグによってシステムやプログラムが作動せず，業務停止等で社会全体に大きな影響を与える恐れが内在しており，システムの品質確保が求められている．一方で，開発の生産性向上も求められており，それは品質保証と相反する要件と言える．これらの顧客要求を満たすには，両者のバランスを極限まで高めなければならない．そのためにはリスク内容を顧客と共有し，コスト認識を持ったうえでテスト方針を確定することで，コスト最適化を図るべきである．

リスクベースドテストは「リスクに対する評価基準を持つこと」「リスク内容を把握し早期に対策を立てること」に焦点が当たることが多いが，何よりも重要なのは「ステークホルダと情報共有を行うこと」である．品質向上については多くの対策を講じているが過剰品質となる傾向があり生産性向上を妨げる要因となっている．また，より大きなリスクの内在を防ぐものではなく，逆に全体品質をも低下させてしまう懸念がある．従来の対策は「テストの品質向上/改善」について深掘りをして対策を行い，障害を防ぐというアプローチに着眼しているが，完全なシステムを構築することは不可能である．リスクベースドテストのように俯瞰的にテスト構造を確認することで必要に応じてテストの省略を許容するというエンジニアリングプロセスの視点が求められると考える．

4.　お　わ　り　に

大規模プロジェクトでの生産性の向上と投入コストの適正化の具体策として，ODIP とリスクベースドテストの適用事例について紹介した．大規模開発の特性として生産性の向上効果は絶対値として大きく，逆に生産性を損なうとその影響も大きい．そのため，生産性向上によりコスト効率を追い求める一方で，行うべきテストについてリスクの可視化と共に安全な効率化を実現することがプロジェクト開発において重要であると認識している．

情報技術は日々変化しているため，今後とも開発ナレッジを積み重ねていくことが重要である．本稿を開発ナレッジの一つとして，コスト効率の良い安全な大規模プロジェクトの実践に向けてご参照いただければ幸いである．

＊1　ODIP：株式会社インテリジェント・モデルが提供する Java ベースの高速開発ツール．
＊2　BankVision：BIPROGY が提供し，地方銀行を中心に 10 行が利用しているオープン勘定系システム．
＊3　モデルドリブン・アーキテクチャ：仕様（モデル図）を基にビジネス要件や，ソフトウェアの機能や構造をプログラムに落とし込んでいく開発手法．ODIP は，本開発手法に基づき，GUI や仕様（モデル図）によって直感的に成果物を自動生成していく開発ツールである．
＊4　MIDMOST：Windows サーバ上で稼働するミッションクリティカルなアプリケーション構築の支援を目的としたトランザクション制御ミドルウェア．
＊5　基準となる「プログラム仕様書（ODIP 版）作成」の標準所要工数は非公表のため，本稿では作業間の比率で表している．
＊6　ユーザや他システムから見える機能の単位（帳票の場合は 1 帳票，他システムインターフェースの場合は 1 ファイル）

＊7　オンライン DB を日付繰越のタイミングでコピーしたバッチ処理用の DB であり，前日基準や前月基準等のサイクルでデータを保持できる仕組みとなっている．

参考文献　［1］　武澤孝弘，樋口英之，“BankVision® 環境に対するプログラムレス開発ツール活用”，ユニシス技報，日本ユニシス，Vol.37 No.2，通巻 133 号，2017 年 9 月，P59 ～ P65.
［2］“ISTQB テスト技術者資格制度　Foundation Level シラバス Version 2018．J03”，日本語翻訳版，JSTQB，2019 年 8 月，P17 ～ 18（1.3 テストの 7 原則）．
http://jstqb.jp/dl/JSTQB-SyllabusFoundation_Version2018.J03.pdf
［3］“実践リスクベーステスト–PRISMA メソッド”，Drs. Erik P.W.M. van Veenendaal CISA”，著/藤原史和訳，2017 年 10 月．
http://www.erikvanveenendaal.nl/site/wp-content/uploads/Practical-Risk-Based-Testing.pdf
［4］“客観的ソフトウェアテスト終了基準の提案”，第 23 年度（2007 年度）ソフトウェア品質管理研究会 第 5 分科会，日本科学技術連盟，2007 年．
https://www.juse.or.jp/sqip/workshop/report/attachs/2007/5_management_report.pdf

※　上記参考文献に記載の URL のリンク先は，2022 年 1 月 20 日時点での存在を確認．

執筆者紹介　寺　尾　和　樹（Kazuki Terao）
2011 年日本ユニシス（株）入社．融資支援パッケージ，BankVision の開発/適用/保守を担当．現在，金融ビジネスサービス第三本部系統システム三部に所属し，BankVision の大規模開発作業に従事．

滝之入　芳　輝（Yoshiteru Takinoiri）
2011 年日本ユニシス（株）入社．TRADEBASE® for FX，MIDMOST の開発/適用/保守業務に従事．現在，金融ビジネスサービス第一本部ビジネスサービス四部に所属し，MIDMOST の開発作業に従事．