派生開発での時間効率性劣化を変更要求から検出する方法

主査　　：飯泉　紀子　（株式会社日立ハイテクノロジーズ）
副主査　　：足立　久美　（株式会社デンソー）
アドバイザー　：清水　吉男　（株式会社システムクリエイツ）
リーダー　　：島崎　稔史　（株式会社インテック）
研究員　　：小笠原　勝　（GE ヘルスケア・ジャパン株式会社）
　　中島　秀人　（東京海上日動システムズ株式会社）
　　中村　直人　（矢崎総業株式会社）
　　中村　奈津子（日本電子株式会社）

研究概要

既存システムへの機能の追加・変更を行う派生開発では，機能の追加・変更により応答時間・処理速度といった性能劣化を招くことがある．これは，性能要求が要求仕様に明記されていないことで性能劣化への配慮不足に陥り易いためである．性能劣化は，開発終盤・納品後に判明することが多く，その改修によって開発コストの増加や納期遅延が発生する．そのため，機能の追加・変更に伴う性能劣化について顧客と交渉が容易な要件定義フェーズで把握する必要がある．そこで我々は，性能劣化を ISO/IEC 25010 で定義されている「時間効率性」の低下と捉え，機能の追加・変更が時間効率性に与える影響を処理時間に換算して検出するための方法「EMOT (Estimation Method Of Time behavior degradation)」を提案する．本手法を過去の不具合事例に適用した結果，時間効率性の劣化を機能の追加・変更要求から検出することができ，要件定義フェーズで性能劣化の対策を講じることで時間効率性の劣化防止に寄与することがわかった．

1. はじめに

派生開発で問題となっている不具合のひとつが，開発終盤のシステムテストや納品以降に見つかる時間効率性の劣化である．派生開発における時間効率性の劣化とは，顧客が期待する応答時間・処理速度ではない場合に「性能が劣化している」と認識される不具合を指す．派生開発では顧客も開発者も機能の追加・変更への対応に注力するあまり，機能の追加・変更が他の機能に与える影響への配慮不足に陥り易い．また，機能の追加・変更において機能との関わり合いが強い「機能要件」の詳細は要求仕様に明記される一方で，機能との関わりが弱い「非機能要件」の詳細は要求仕様に明記されないことが多い．しかし，「明確に要求していないのだから現状と同じであることが当然」と顧客が期待しているため，機能の追加・変更に伴う振る舞いの変化が時間効率性の劣化と判断される要因となっている．

我々は，派生開発における時間効率性の劣化による過去の不具合事例を調査･分析した．併せて，不具合の詳細を正確に把握するために不具合の改修を行った開発者への聞き取りを実施した．その結果，開発の終盤やシステムテストで判明する時間効率性の劣化に関して以下に示す 3 つの特徴に分類できることがわかった．

1) 時間効率性が変化する認識を持っていない．
2) 時間効率性が変化する認識は持っているが，変化を劣化とは考えていない．
3) 顧客に時間効率性が劣化することを説明出来ていない．

そこで，開発者が時間効率性の変化を具体的な劣化として把握することができ，同時に設計段階で時間効率性の劣化に気が付くことができれば，手戻りによる開発コストの増加

や納期遅延の発生を防止できると考えた．そのため，機能の追加・変更による振る舞いの変化が現状の時間効率性に与える影響を処理時間に換算して具体的な影響を検出する方法「EMOT」を提案する．

以降，2 章で現状の時間効率性の劣化による問題を検出することができなかった過去の不具合事例を分析した結果を示す．併せて，時間効率性の劣化が発生する原因および時間効率性の劣化に関する先行研究の有無を示す．3 章では，解決策として機能の追加・変更が時間効率性におよぼす具体的な影響を検出するための方法を提案する．4 章では，検出することができなかった過去の不具合事例に対して，提案する解決策を適用することで時間効率性の劣化を早期に判明させられるかを検証する．5 章はまとめと今後の課題である．

2. 現状分析

2.1 時間効率性の劣化とは

時間効率性とは，ISO/IEC 25010 において「製品又はシステムの機能を実行するとき，製品又はシステムの応答時間及び処理時間，並びにスループット速度が要求事項を満足する度合い」と定義されている．

そして，時間効率性の劣化とは，すでに顧客に受け入れられ，認知されている現状の時間効率性に変化が生じ，その変化が現状のシステムと比較して顧客の期待する結果から乖離している場合に「使いづらくなった」「操作応答を待っている間の待機時間が多くなり作業効率が低下した」と顧客が認識した状態を指す．具体的な例を図 1 に示す．

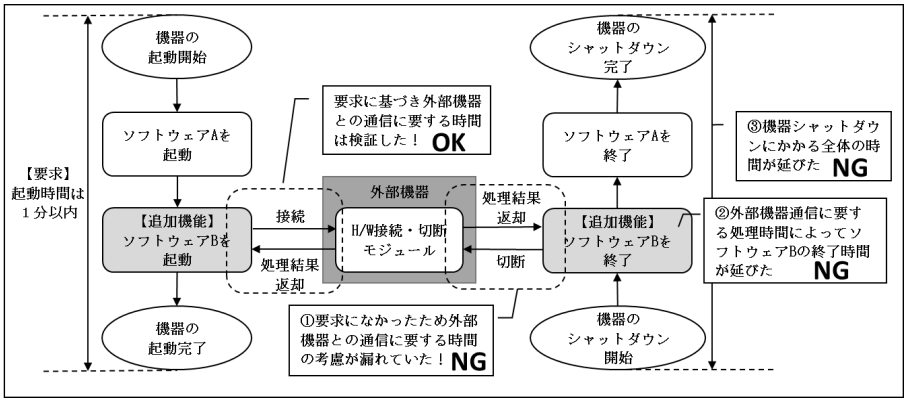

図 1　時間効率性が劣化しているという指摘の例

この例は，「機能要件に応じて制御を実行するためのソフトウェア B を追加した．ソフトウェア B の追加に伴い，顧客から要求があった起動時間については単体テストの時点で要求を満たしていることを確認したが，要求がなかった終了時間については確認しなかった．その結果，対象のソフトウェア B を搭載した機器のシャットダウン時間が既存の機器と比較して遅くなっており，機器を頻繁に移動する必要がある顧客はその都度機器の起動・終了を行うため，業務を阻害してしまう」という状態である．

2.2 時間効率性の劣化による不具合の分析

派生開発でどのような時間効率性の劣化が発生しているのかを把握するために，過去の不具合事例 20 件を収集して分析した（図 2）．

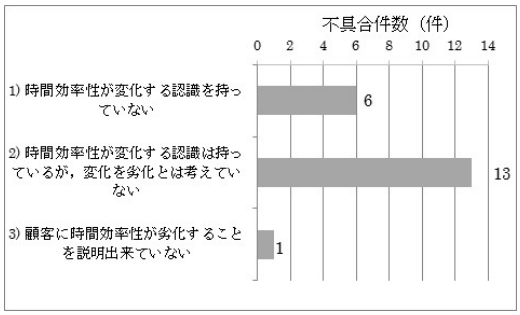

図 2　不具合事例件数と特徴の内訳

これらの不具合を分析した結果，以下に示す 3 つの特徴に分類できることが分かった．

1) 時間効率性が変化する認識を持っていない
機能の追加・変更に注力するあまり，時間効率性の変化が数値など目に見える形で表れないと，それを「劣化」とは認識できないという時間効率性の変化に対する意識の低さを指す．また，短納期・少人数という制約を受ける開発体制ではこの意識の低さがより顕著である．

2) 時間効率性の変化を劣化とは考えていない
機能の追加・変更に伴い時間効率性に変化が生じる事は漠然と認識しているが，その変化が「劣化」として機能の追加・変更に影響するのかは認識していない状態，または変化する時間を具体的な数値として把握出来ていないことを指す．
3) 顧客に時間効率性が劣化することを説明出来ない
ソフトウェア設計など開発の早い段階で顧客に対して機能の追加・変更要求が時間効率性の劣化を招くことを具体的な数値で説明できていない．そのため，劣化に対する代替案を検討することの有用性を顧客に訴求することが出来ないことを指す．

以上のことから，派生開発での機能の追加・変更に伴う時間効率性の影響に対する開発者の配慮不足が，時間効率性の劣化が判明する時期を遅らせる要因であると言える．

そこで我々は，顧客および開発者に対して機能の追加・変更要求に伴う時間効率性の劣化の可能性を目に見える形で示すために必要となる「時間効率性の劣化を要件定義フェーズで具体的な数値で示す」ことを課題とした．この課題を解決し，顧客の「明確に要求していないのだから現状と同じであることが当然」という暗黙の期待に応えられる様にする．

2.3 先行研究

時間効率性の劣化に対して，先行研究で解決することができるか否かを調査した．以下に調査結果を示す．

派生開発の要求に対し，必要不可欠なプロセス・成果物で構成され，短納期という状況に対応した手法 XDDP(eXtreme Derivative Development Process)[1]がある．この手法は，テスト工程での欠陥抽出および修正工数を削減する効果を得るために，変更要求仕様書，トレーサビリティ・マトリクスおよび，変更設計書の 3 点セットの作成と，それを基としたレビューの実施により影響範囲の検討漏れを検出し易くするよう工夫されている．しかし，我々が課題とした「時間効率性の劣化を具体的な数値で示す」ために必要な「時間効率性の変化有無とその影響を調査」する具体的な方法については明記されていない．

次に，機能の追加・変更による影響の相関を機能単位でマトリクスとして表現する「影響マトリクス」を用いてテストの漏れや影響の絞りこみを行なう手法[2]がある．この手法は，XDDP に影響マトリクスを使用するプロセスを追加することで表面化した影響を XDDP で見直すように工夫されている．しかし，この手法は表面化した影響を顧客に許容可能か否かを確認するためのプロセスが提案されておらず，開発終盤・納品後に影響が判明する可能性がある．

最後に，ベース・システムの時間，領域の限界値を超える変化を「間接リソース変化点」として定義し，機能の追加・変更に伴う影響を「リソース」という影響範囲の特定が容易な観点で捉え，変更漏れによる不具合や機能の追加・変更による機能性の低下を防止する手法[3]がある．この手法は，間接リソース変化点を用いて詳細が要求仕様に明記されていない要求を検出した上で，対策することを可能としている．また，具体的な数値が可視化されるまで機能の追加・変更による影響範囲を特定することを遅らせることにより，手法の精度を高めている．この手法の精度を維持したまま，影響範囲を特定する時期に対する依存を低減することができれば，我々が課題としている「時間効率性の劣化を具体的な数値で示す」ことが可能と考える．

以上の結果から，「間接リソース変化点」の観点を用いた上で「顧客と機能の追加・変更に関する交渉が容易である要件定義フェーズで時間効率性の劣化を検出する」ことができる解決策を検討することとした．

3. 解決策

3.1 解決の方針

先行研究の調査結果から，顧客の「明確に要求していないのだから現状と同じであるこ

とが当然」という暗黙の期待に応えるべく，「時間効率性の劣化を具体的な数値で示す」という解決策を検討することとした．以下に，その方針を示す．

Step1)　時間効率性の変化を認識し易くする

Step2)　時間効率性の変化を定量的に示して変更箇所を可視化する

そして，Step1 および Step2 にて定量的に示し，可視化した情報から時間効率性の変化量を把握し，顧客との交渉や設計見直しの実施等を行なう段階を Step3 とした．

3.2　EMOT (Estimation Method Of Time behavior degradation)

3.1 で述べた通り，顧客の「明確に要求していないのだから現状と同じであることが当然」という暗黙の期待に応えられる様にするための解決策として，時間効率性の変化を認識し易くするための観点をまとめた「改訂版リソース変化点確認表」を作成した．また，時間効率性の変化を定量的に示して変更箇所を可視化するために「時間効率性記入表」を考案した．これら 2 つの表から導出した時間効率性の変化量に基づいて顧客との交渉や設計見直しのきっかけを開発者に与える手法を「EMOT (Estimation Method Of Time behavior degradation)」と定義した(付録 D)．

以降，改訂版リソース変化点確認表の詳細に関しては 3.2.1 に，時間効率性記入表の詳細に関しては 3.2.2 に示す．

3.2.1　改訂版リソース変化点確認表

時間効率性の劣化に対する知見や経験を有していない開発者が時間効率性に対する影響を検討しても，時間効率性の劣化に繋がる観点の見落としに気づかなかったり，検討すべき観点を誤り「考慮した」と判断する「思い込み」が生じたりする恐れがある．したがって，開発者が考慮すべき時間効率性の劣化に繋がる観点を示す必要があると考えた．

そこで我々は，先行研究[3]で定義されている「リソース変化点チェックシート」を基に「改訂版リソース変化点確認表」を考案した（表 1）．

表 1　改訂版リソース変化点確認表

No	分類	リソース変化点チェック項目	時間効率性の劣化目安
1	変数・配列	配列のサイズに変化はないか	
2	メモリ	確保するメモリサイズに変化はないか	
3		同時に確保されるメモリサイズに変化はないか	
4		メモリへの書込みサイズに変化はないか	±15ﾐﾘ秒
5		ハンドルの生成数に変化はないか	
6		同時に生成されるハンドルの数に変化はないか	
7		スタックの消費に変化はないか	
8		メモリマップが変更となる変化はないか	
9	処理	関数の処理時間（returnするまでの時間）に変化はないか	
10		再帰関数，循環関数が追加されていないか	
11		処理回数に変化はないか	
12		while文，for文のループ回数に変化はないか	
13		応答・通知を返却する時間に変化はないか	
14		同期，非同期処理に変更はないか	
15	データ	扱うデータサイズに変更はないか	
16	ディスク	ディスクI/Oに変化はないか	
17		ディスクI/Oの優先順位に変化はないか	
18		ディスクI/Oの応答時間に変化はないか	
19	ネットワーク	ネットワークI/Oに変化はないか	
20		ネットワーク使用率に変化はないか	

この表は,「リソース変化点チェックシート」に時間効率性の劣化に繋がるリソース変化点を記入する列である「時間効率性の劣化目安」を追加した表である．

「時間効率性の劣化目安」は，機能の追加・変更が時間効率性に与える影響を考慮するための指標であり，開発者が機能の追加・変更の中から時間効率性の劣化に繋がるリソース変化点を抽出し，その具体的な影響を考慮する際に使用する観点である．

「時間効率性の目安」に記入する目安時間は,「EMOT」を適用するシステムごとに決定する．例えば，過去の派生開発でメモリへの書き込みサイズに変化が生じると,その度合いによらず時間効率性が 15 ﾐﾘ秒変化することがすでに判っている場合，No4 の「メモリへの書き込みサイズに変化はないか」の「時間効率性の劣化目安」には±15 ﾐﾘ秒と記入する．このように，劣化目安の値を類推するのではなく，過去の派生開発の実績を引用することで数値の精度を高めることができる.「時間効率性の劣化目安」でマイナス値を考慮する理由は，他機能や通信の処理タイミングなどの問題でレスポンスが早くなることが問題となる場合や，現状の時間に慣れてい

る顧客にとって応答時間が早くなることが，遅くなることと同様に違和感を覚え，その結果，不具合と指摘される可能性があるためである．

3.2.2 時間効率性記入表

機能の追加・変更が時間効率性に及ぼす具体的な影響を検出するため,「時間効率性記入表」を定義した（表 2)．以下に，時間効率性記入表の使用手順を示す．

表 2　時間効率性記入表

変更要求	顧客の操作	チェック項目No	現状の応答時間（ミリ秒）	予測時間（ミリ秒）	予測変化率（%増減）	実績（ミリ秒）
(例)検索条件を3つから4つに変更してほしい	検索条件を選択する					
	検索ボタンをクリックする					

手順 1：変更要求の記入

機能の追加・変更の要求を「変更要求」欄に記入する．この際，プロジェクトによっては変更要求が多く,その全てに本手法を適用することが困難な場合がある.そのため,「変更要求」欄に記載する変更要求は，重要度をフィルターにして記入の是非を決定する．フィルターとは,「EMOT」を適用するプロジェクトにおいて重要視される観点のことであり,「EMOT」の適用前にプロジェクト毎に予め作成する必要がある．例えば，過去の派生開発にて「検索条件を追加したら，応答時間が 3 倍遅くなった」という事例があったとする．これを表 1 のリソース変化点チェック項目にある No16 に該当するものとし，フィルターの項目の 1 つとする．その上で，今回の派生開発の変更要求の中に No16 に関連する変更要求が含まれているか否かを確認する．もし，含まれている場合は，過去の事例と同様の時間効率性の劣化が発生する可能性があるため，時間効率性記入表の変更要求欄に記載し，以降の手順に沿って時間効率性の変化を確認する．

手順 2：顧客の操作の記入

開発者は，機能の追加・変更対象となる機能および処理を顧客の操作手順を基に分割し，それを「顧客の操作」欄に記載する．操作手順が不明な場合は，必要に応じて顧客に確認を取り,「顧客の操作」欄に記入する．操作手順で分割する理由は，分割の単位をプロジェクトメンバーと共有し易くするためである．また，開発者の知見や経験の差による分割の単位のばらつきをできるだけ無くすために，分割した「顧客の操作」はプロジェクトの有識者がレビューを行なうこととする．加えて，顧客は時間効率性の変化を「劣化」とみなすため，顧客視点の操作手順を基とすることで，時間効率性の劣化に対して開発者が気付きを得る効果がある．

手順 3：改訂版リソース変化点チェック項目の記入

手順 2 で分割した操作手順ごとに,時間効率性の劣化に繋がるリソース変化点の有無を判断する．この判断には，改訂版リソース変化点確認表を使用する．チェック項目に該当する場合は，その番号を「チェック項目 No」欄に記載する．なお，改訂版リソース変化点確認表の内容については,本手法を適用するソフトウェアの分野に応じた適切なチェック項目をあらかじめ準備しておく．また，開発の進捗に応じ，追加・変更および削除すべきチェック項目が発生した場合は，適宜一覧を更新する．

手順 4：現状の応答時間の記入

手順 2 で分割した操作手順ごとに，現状の応答時間・処理速度を調査し，その結果を「現状の応答時間」に記入する．調査例として，サーバーのログからトランザクションの開始から終了までに要する応答時間・処理速度を分割した顧客の操作の単位で収集し,その結果を記入する．顧客の操作の単位で応答時間・処理速度を算出することが出来ない場合は，トランザクションの開始から終了までの合算値を記入する．

手順 5：予測時間・予測変化率の記入

手順 4 で調査した現状の応答時間・処理速度に対して，改訂版リソース変化点確認表

の「時間効率性の目安」を参照し，各操作手順に該当するリソース変化点の応答時間・処理速度を計算した結果を「予測時間」に記入する．このとき，システム上に類似機能が存在する場合や過去プロジェクトで実施した機能の追加・変更で既に収集済みの時間効率性の目安がある場合，それらを基に予測時間を計算することができる．しかし，類似機能や収集済みの時間効率性の目安が無い場合，追加・変更する処理量やデータ量を見積もり，現状の応答時間に変化率を掛け合わせるなどして予測時間を計算する必要がある．「時間効率性の目安」についてもチェック項目同様に，本手法を適用するプロジェクトごとに決定する必要がある．なお，「予測変化率」には，予測時間から現状の応答時間を差し引いた値を現状の応答時間で割るよう計算式を記入しておく．「予測変化率」を割合として記入する理由は，現状の応答時間と予測時間の差が小さくても割合が大きいと問題になることがあるためである．

手順 6：実績時間の記入

開発の終了後に，開発によって実際に変化した応答時間・処理時間を「実績」欄に記入する．これにより，予測時間と実績の差異が確認できるため，次回の変更時の参考値として扱うことができる．また，その差異を基に改訂版リソース変化点確認表の「時間効率性の目安」を見直すことにより，予測時間の精度を高めていくことができる．

4. 解決策の検証

4.1 検証内容

3.1 で述べた Step1（時間効率性の変化を認識し易くする）および Step2（時間効率性の変化を定量的に示して変更箇所を可視化する）が有効であるかを判断するべく，時間効率性の変化が劣化として問題となった過去の不具合事例を用いて，下記について検証した．

1) 時間効率性の変化を認識できるか
2) 導出した予測時間が実績と同程度か
3) EMOT の適用工数

ただし，今回の検証では 3.1 で述べた Step3 の検証は実施できていないため，2.2 で述べた顧客の暗黙の期待に応えるという課題を解決出来るか否かを検証するに至っていない．

4.2 検証結果

検証結果を表 3 に示す．なお，今回は業態の異なる 4 社において合計 7 件の不具合事例にて検証したが，手順 5 で説明した予測値の算出例を示すために 3 件の事例を掲載した．

表 3　検証結果

変更要求	顧客の操作	チェック項目 No	現状の応答時間 (ミリ秒)	予測時間 (ミリ秒)	予測変化率 (%増減)	実績 (ミリ秒)
変更要求 1 新規機能のために必要となる外部機器の起動とシャットダウンの制御を行う．外部機器は本体機器の起動とシャットダウンと連動して制御する．	本体機器を起動する	No.9 No.13	120000	120000	0	120740
	本体機器をシャットダウンする	No.9 No.13	25000	45000	80	46500
変更要求 2 自動調整のために信号を取得する際に，現状ではハードウェアのリセット無しに行っている．信号取得のつどハードウェアのリセット操作を行うことにより精度が上がるようにする．	自動調整を開始する	No.13	10000	22100	121	25000
変更要求 3 現状では初期設定に使用するファイルが専用フォーマットとなっている．これをエクセルファイルに変更する．	対象のファイルを指定する	-	-	-	-	-
	読み込みボタンをクリックする	No.9 No.15	800	14400	1700	80000

検証対象の事例に含まれる「改訂版リソース変化点確認表」の項目について，予測時間を下記のように算出した（表 4）.

表 4　予測値算出の経緯

変更要求 1	外部機器をシャットダウンするために必要な時間が最小で 20000 ﾐﾘ秒である. シャットダウンに必要な通信は数バイトのデータを一度だけ送信するため，データ通信に要する時間は考慮せず，外部機器のシャットダウンに要する待ち時間 20000 ﾐﾘ秒のみが増加すると予想した.
変更要求 2	ハードウェアの制約上リセット一回の処理時間が最大 1100 ﾐﾘ秒となる．11 枚撮影のオートフォーカスで撮影ごとにリセットを実施するため処理時間が最大 12100 ﾐﾘ秒増加すると予測した.
変更要求 3	データ量が 1.5 倍，処理量が 12 倍程度となるため，現状の応答時間 800 ﾐﾘ秒に 1.5×12 をかけた 14400 ﾐﾘ秒を予測時間とした.

7 件の事例に対して合計 5 名の開発者がそれぞれ 1～2 件ずつ検証を行った．自分が開発に従事したプロジェクトで発生した不具合事例を検証したのは 2 名，件数は 2 件である．残りの 3 名が検証した 5 件は，自分が開発に従事していないプロジェクトで発生した不具合事例である.

検証の結果，表 3 以外の事例を含めると，7 件中 7 件の事例にて時間効率性の変化を認識し，変更箇所を可視化することができた．また，予測時間と実績値を比較したところ，3 件の事例にて実績の±20%以内に予測時間が収まるという高精度な予測ができ，時間効率性の変化が大きいことを予測できたものの実績値との乖離が大きかった事例が 2 件，予測変化率が低かった事例が 2 件であった.

なお，変更要求 1 は 2.1 で述べた図 1 の事例である．表 3 で示した通り，変更要求を顧客の操作に分割したことにより変更要求に存在しなかったシャットダウンの時間を可視化することができたため，時間効率性の劣化に気付くことができた.

4.3　考察

今回，業態の異なる 4 社での過去の不具合事例について「EMOT」を適用した．4.2 に示した通り，いくつかの事例で時間効率性の劣化を検出できた．検出できた事例は，表 3 の変更要求 1 および変更要求 2 のような処理すべきデータ量の見積もりが容易かつ，処理がシーケンシャルな事例であった．このような事例においては，「EMOT」で時間効率性の劣化を検出し易く，予測値の精度も高くなることがわかった.

一方で，時間効率性の劣化を検出できない事例や予測値と実績値が大きく乖離した事例もあった．これらの事例には，組み込み系システムにおけるスレッドの同期待ちで予測しがたい遅延を生じたもの，データ処理量の増大によるメモリの負荷が処理時間の大幅な増加の原因となったものなどがあった．これらはいずれも変更されるデータなどの何らかの測定値から時間効率性への影響を線形に対応づけることが困難である. このような場合は，「EMOT」では有為な効果を得られないことが推測される.

また，検証で得られた「EMOT」の適用工数は 1 つの事例あたり 3～4 時間程度であった（過去の不具合事例での検証のため，フィルターの作成工数ならびに改訂版リソース変化点確認表の作成工数は除外した）. なお，検証は当該システムの開発経験が 2 年以上の者が行なったものである．開発経験が少ない場合，顧客の操作の分割や予測時間の算出に工数を要する可能性があるが，開発経験が豊富な開発者であれば工数は削減できると考える.

不具合対応工数と「EMOT」の適用工数を比較すると，表 3 の変更要求 2 については不具合対応に 42.1 時間費やしており，「EMOT」の適用工数の方が少ないという結果となった．ただし，現場での「EMOT」の運用を考慮すると，適用には，前述の手順 1 で述べたように全件実施するのではなく，フィルターを通して必要と思われる変更要求を選定して実施することで「EMOT」を作成する負荷を減らし定着させることが重要と考える.

また，「EMOT」について開発者に説明を実施し，実運用面での有用性をヒアリングした．その結果，「具体的な数値を導出できる強みを活かし，顧客と交渉する場面で使用したい」との回答を得た．さらに，表 3 の変更要求 3 のような予測の精度が低い場合でも，「時間効率性記入表を作成することで時間効率性への影響を意識するようになった」との回答を得

た．これらの結果から，「機能の追加・変更への対応に注力するあまり，他の機能に与える影響への配慮不足に陥り易い」という，現状の派生開発での問題の改善ならびに 3.1 で述べた Step3 である顧客の説得や設計の見直しに繋げられると期待できる．

5. まとめ

5.1 結論

顧客の「明確に要求していないのだから現状と同じであることが当然」という期待に応えるべく，顧客および開発者に対して機能の追加・変更要求に伴う時間効率性に劣化が生じる可能性を目に見える形で示すために必要となる時間効率性の劣化を具体的な数値で示す手法「EMOT」を提案した．

「EMOT」は，「時間効率性記入表」に記入した変更要求を「時間効率性の劣化が表面化する顧客の一連の業務」に基づき顧客の操作に分割し，「改訂版リソース変化点確認表」に該当するチェック項目を選択した上で時間効率性への具体的な影響を顧客の操作単位で導出できるよう工夫している．この「EMOT」を過去の不具合事例に適用したところ，7 件中 3 件の事例で開発者が時間効率性への具体的な影響を確認することができた．

昨今の派生開発は，短納期・少人数での開発を要求される状況が多い．また，新規開発時の開発者が居ないなどの理由により当時の状況や背景の把握が困難な場合も想定される．これらの理由から機能の追加・変更がもたらす時間効率性への具体的な影響を全て把握することは困難である．しかし，「EMOT」を活用することで，時間効率性の具体的な影響の程度の把握に大いに寄与できると考える．

5.2 今後の課題

「EMOT」は過去の不具合事例では有用性を確認することができた．今後の課題は以下 2 点を挙げる．

1) 「EMOT」の改善
   「EMOT」ではデータ量や処理に比例するケースでは時間効率性の劣化を検出できる可能性が高いが，ある境界を超えた場合に急激に処理時間が増加する事象など，データ量や処理に比例しないケースについては検知することが難しいと考えるため，「EMOT」の更なる改善が必要である．
2) 実際のプロジェクトに適用する
   今回の研究では過去事例に適用したが，今後は実際のプロジェクトに適用し，有用性の確認および運用上の問題点を明らかにする．また，検証するに至らなかった実際に顧客を交えて問題を未然に防止する効果について確認していく．適用していく中で，時間効率性記入表を作成するための改訂版リソース変化点確認表の作成については，より汎用的な分類や不具合が発生し易いユースケースにも着目し，現実の問題解決を通して新たなチェック項目の追加を検討していく．このチェック項目のバリエーションや使い方を体系的に整理すれば，さらに大きな効果が期待できる．

上記の課題に継続して取り組むことで「EMOT」の更なる効果が期待できる．

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，2005
[2] 奥山剛，奥田享一郎，吉本吾朗，永田敦，中森博晃，村上聡，丸山久，柳内政宏，派生開発におけるモレ・ムダのないテスト設計，日本科学技術連盟 SQiP 研究会分科会報告書，2009
[3] 小瀬聡幸，衣斐省伍，井貝智行，杉山幸雄，XDDP の変更設計書から間接リソース変化点を抽出する手法，日本科学技術連盟 SQiP 研究会分科会報告書，2013