2004－SE－146　(4)
2004／11／26

設計･ソースコードを対象とした個人レビュー手法の比較実験

野 中　　誠†

個人レビューの効率性および有効性を改善することは，ソフトウェア製品の品質および生産性を向上させるために重要である．個人レビューの効率性および有効性は，レビュー手法だけでなく，除去対象欠陥の型，成果物の特性，個人のスキルなどによって異なる．どのレビュー手法が，どのような状況において効率および効果を発揮するのかを把握することが求められる．本稿では，個人によるソフトウェア開発作業における設計成果物およびソースコードを対象に，Test Case Based Reading (TCBR) と Checklist-Based Reading (CBR) の 2 つのレビュー手法の比較実験を行った結果について述べる．実験の結果，TCBR の方が効率性および有効性が高く，摘出できる欠陥型の傾向に違いがあることが本実験の範囲で確認できた．

An Experimental Comparison of Individual Review Methods for Design and Source Code

MAKOTO NONAKA

Improving efficiency and effectiveness of individual review is important to enhance software product quality and project productivity. The efficiency and effectiveness of individual review depend on not only the reading method a developer applies but also defect types, artifact characteristics, or his or her skill. It is important to understand the context in which a reading method has its effectiveness and efficiency. This paper describes an experiment that evaluates the efficiency and effectiveness of two reading methods, Test Case Based Reading (TCBR) and Checklist-Based Reading (CBR). These methods are intended to be used in design review and code review phase. The result showed that TCBR was more effective and efficient than CBR in the context of this experiment.

1. はじめに

ソフトウェアレビューは人手によりソフトウェア成果物を確認して欠陥を除去するプロセスである．レビューは, その公式さ度合いによって, パスアラウンド，ウォークスルー，チームレビュー，インスペクションなどに分類される[1]．とくにインスペクションなどの公式さ度合いの高いレビューは品質向上の効果が大きく[2]，ソフトウェア開発における良いプラクティスのひとつとされている．しかし，そもそもインスペクション対象の成果物の品質が悪いと，インスペクションの効率が低下し，欠陥の見逃しも多くなるため，プロジェクト全体の生産性および品質低下を招く．また，インスペクションにはある程度の工数を必要とするため，開発期間やコストなどの制約のために，すべての成果物に対してインスペクションを実施できない場合が多い．したがって，開発者自らが個人レビュー[3]を実施するなどして，成果物の品質を十分に確保することが求められる．

ソフトウェアレビューの効率性および有効性は，採用したレビュー手法，除去対象欠陥の型，成果物の特性，個人のスキルなどによって異なる．レビュー手法には，無計画に実施される Ad Hoc Reading，チェックリストを用いた Checklist-Based Reading (CBR)[2], 系統的な手法である Scenario-Based Reading (SBR) などがある[4]．近年，CBR と SBR との比較研究などが行われており[4-8]， SBR の方が CBR よりも効率性および有効性において優れているという結果が示されている．しかし，手法が適用された状況によっては逆の結果も示されており，手法の適用範囲や有効性を発揮するコンテキストを明らかにするなど，一層の研究が求められる．また，これらの研究の多くは上流工程におけるレビューを対象としたものであり，下流工程における個人レビューを対象としたものは少ない．

一方で，eXtreme Programming (XP) [9]で採用されているテストファーストのプラクティス，すなわち，開発作業の前にテストケースを作成して常に自動テストが可能な状態にしておくことも，品質確保のための有効な手法である．XP では，テストファーストの実施にあたって，jUnit[10]などのツール利用を想定している. ただし, ツール利用を想定しない場合であっても，テストケースを作成してから開発を行うことは，有効

†東洋大学 経営学部
Faculty of Business Administration, Toyo University

性の高い手法である．

本研究では, 開発作業の前にテストケースを作成し，それに基づいて成果物をレビューする手法を Test Case Based Reading (TCBR) と呼ぶ．本研究では，個人のソフトウェア開発作業における設計レビューおよびソースコードレビューを対象に, TCBR および CBR がどのような状況で効率性および有効性を発揮できるのかを明らかにすることを目的としている．本稿では，その試みとして，ある企業の新人向けソフトウェア技術研修において実施した簡易版 PSP (Personal Software Process) 演習[3]を対象に, TCBR と CBR の 2 つのレビュー手法の比較実験を行った結果について報告する．

2. レビュー手法

2.1 Checklist-Based Reading (CBR)

CBR は従来から利用されている典型的なレビュー手法であり[2]，Ad Hoc Reading とともにもっとも広く利用されている手法である[11]．CBR では，過去の経験などに基づいて作成されたチェックリストを用いて，成果物に欠陥が含まれていないかを確認する．一般に，複数のチェック項目を同時に考慮すると欠陥の見落としが発生しやすくなる．そのため，モジュールなど成果物の構成要素に対してチェック項目を一つずつ適用し，これをすべての構成要素について順に行うことで網羅的にレビューする方法がとられる[3].

CBR はすべてのチェック項目に関して網羅的にレビューできる反面，レビュー時間が長くなってしまうという欠点がある．また，実務における現実の課題として，経験の蓄積に伴ってチェック項目が膨大になってしまい，すべてのチェック項目を人手でレビューするのが容易でなくなる場合がある．したがって，ツールによるチェックの自動化，設計技法の改善によるチェック項目の不要化，レビュー手法の効率化などが求められる．

2.2 Scenario-Based Reading (SBR)

SBR は, レビュー実施時におけるシナリオ[1]に基づいて成果物をレビューする手法である．SBR には，利用者・設計者・テスト担当者などの観点別に着目すべき事項を明示してレビューを実施する Perspective-Based Reading (PBR)[12][13], ユースケース記述など利用者とシステムとの対話手順に基づいてレビューを行う Usage-Based Reading (UBR)[7]，欠陥型に基づいてレビューを行う Defect-Based Reading (DBR)[14]などの手法がある[4]．PBR や UBR は，主に要求仕様や設計仕様など上流工程での成果物に対して適用されるレビュー手法である．

文献[11]によれば，SBR は CBR よりも系統的な手法として位置づけられている. いくつかの研究では, CBR よりも SBR の方が効率性および有効性に関して優れていることが示されている[4-8]．しかし，SBR が常に有効であるわけではない．さらには，Basili らの研究では，レビューよりも機能テストの方が有効な場合もあると報告されている．

2.3 Test Case Based Reading (TCBR)

本研究では，テストケースに基づいて成果物をレビューする手法を TCBR と呼ぶ．TCBR は，主に設計工程以降の工程における成果物を対象としたレビューで適用されることを想定している．テストケースごとに成果物をレビューする際に，あらかじめ用意した欠陥型（例えば文献[3]の欠陥型）を参照しながら，レビュー対象の成果物に欠陥がないかをチェックする．TCBR は，UBR と DBR を融合させたレビュー手法であるといえる．

TCBR を実施するためには，テストケースをあらかじめ用意しなければならない．そのため，開発作業を開始する前に，開発者個人に割り当てられたコンポーネントの要求仕様に基づいて，ブラックボックステスト用のテストケースを作成する．テストケースの作成には, 一般的な手法である同値分割技法が適用できる．また，テストケースの表現には図 1 のような決定表を用いると見やすくてよい．

チェック条件／確認内容		テストケースNo.	1	2	3	4	5	6	7	8	9	10	11
チェック条件	テストデータ	右記データファイル	○	○	○	○	○	○		○	○	○	○
		存在しないファイル名							○				
	μ0	187	○	○	○						○	○	○
		194				○	○						
		190.5						○					
		"abc"								○			
	σ	3	○	○	○	○	○	○				○	○
		"abc"									○		
	検定	1（片側検定：μ＜μ0）		○		○							
		2（片側検定：μ＞μ0）	○										
		3（両側検定：μ≠μ0）			○		○	○					
		4										○	
		"abc"											○
確認内容	P値 ＝ 0.00982		X			X							
	P値 ＝ 0.01963				X		X						
	P値 ＝ 0.99018			X									
	P値 ＝ 1.00000							X					
	検定統計量 ＝ 2.33		X	X	X								
	検定統計量 ＝ -2.33					X	X						
	検定統計量 ＝ 0.00							X					
	エラーメッセージを出力し、プログラムを終了する。								X	X	X	X	X

図 1　決定表によるテストケース表現

ただし，実際の開発の場面では，テストケース数が膨大になる場合が少なくない．そのため，すべてのテストケースについてレビューを行うと，レビューに費

[1] ここでのシナリオとは，ユースケース記述などの操作シナリオとは異なり，レビュー実施時のシナリオを意味する．

やす工数が大きくなってしまう．したがって，レビュー対象の成果物規模を小さくする，直交表を利用してテストケースを縮約する，同一のフローグラフを通過するテストケースは省略または簡略化するなど，レビューの効率化を図る必要がある．

以下に，TCBR の実施手順を示す．

Step 1:詳細設計を行う前に，コンポーネントの要求仕様に基づいてテストケースを作成する．テストケースの作成には，同値分割技法を用いるなどして，無駄のない十分なテスト項目を作成する．

Step 2: 作成したテストケースについて, 正常系のテストケースから順番に，設計成果物またはソースコードの実行フローを追跡してレビューする．ただし，複雑な計算式や繰返し処理などの部分は，具体的な数値を用いた試算を行わずに，計算式や実行フローの論理的な整合性のみをチェックする．その理由は，このようなチェックは適切なテストケースを用意して機能テストを実施した方がレビューよりも効率がよい場合が多いためである．なお，ソースコードは，新規・変更行のみをレビュー対象として，再利用コードはレビュー対象としない．

Step 3: 設計成果物またはソースコードの追跡時に, 用意した欠陥型に基づいて欠陥がないかをチェックする．確認が終わった部分には，条件分岐や繰返し処理部分を除き，確認済み記号をつける．確認済みの部分は，それ以降のレビュー対象から除外する．

Step 4: すべての正常系テストケースについてチェックが完了したら，異常系テストケースについて同様のレビューを行う．すべてのテストケースについてチェックが完了したらレビューを終える．

3. レビュー手法の比較実験

3.1 概要

TCBR と CBR の効率性および有効性を比較する目的で，レビュー手法の比較実験を行った．実験は，ある企業の新人向けソフトウェア技術研修のうち，2 日間をかけて実施した簡易版 PSP 演習[3]のなかで実施した．被験者は，C 言語の基礎的な教育を終えた新人ソフトウェア技術者または電子工学技術者 33 名である．いずれも理工系大学または大学院の卒業生または修了生であり，プログラミング経験の浅い人から経験豊富な人まで様々であった．これらの被験者を，後述する方法で 2 つのグループに分け, それぞれ CBR と TCBR による個人レビューの比較を行った．

3.2 実験目的

本実験の目的は，開発者個人によるプログラムコンポーネント開発において，次の点について TCBR と CBR を比較することである．

(1) 効率性
(2) 有効性
(3) 発見しやすい欠陥型

なお，本実験におけるプログラムコンポーネントとは，一人の開発者に開発が割り当てられたプログラム単位であり，複数の関数モジュールから構成されるプログラム単位を意味する．本実験では，半日あれば十分に作成できる程度のプログラム規模とした．

3.3 基準値の測定と被験者のグループ分け

レビューの効率性および有効性を定量的に評価するために，本実験では，被験者にプロセスデータの測定を行ってもらう．そのため，被験者にはプロセス測定の作業にある程度慣れてもらっておく必要がある．また，個人レビューを導入した前後のパフォーマンス比較を行ったり，被験者をパフォーマンス別にグループ分けしたりするために，各被験者の開発パフォーマンスの基準値を測定しておく必要がある．

被験者によるプロセス測定を少しでも容易にするために, 本実験では PSP0 プロセス[3]を簡略化した sPSP0 (Simplified PSP0) プロセスを定義した．また，PSP0 用のプロセスデータ記録帳票を sPSP0 用に修正し, Excel 上で入力できるようにした．これらを用いて，被験者には 3.4 節で述べるプログラミング課題 1C に取り組んでもらい，そのプロセスデータを被験者自身に測定してもらった．

TCBR と CBR を独立に評価するため, 被験者を均等な 2 つのグループに分ける必要がある．本実験では，課題 1C 作成時に sPSP0 により測定された基準値データのうち，単体テストおよび受入れテストにおける除去欠陥の欠陥密度順に被験者を並べ，欠陥密度に関して両グループが均等になるように被験者を分けた．以降では，それぞれのグループを TCBR グループおよび CBR グループと呼ぶ．

3.4 プログラミング課題

PSP のプログラミング課題 5A[3]をもとに要求仕様およびテストケースを詳細化し，プログラミング課題 1C および 2C を作成した．その概要を以下に示す．

1C：シンプソン法によって任意の関数を数値積分するプログラムを作成する. 積分の対象とする関数には,

標準正規分布の関数を使用する．

2C：テキストファイルから数値データを読み込み，母平均の差に関する検定（母分散既知の場合）を行うプログラムを作成する．

被験者は，これらの課題を C 言語でプログラミングする．課題の要求仕様はいずれも 2 ページであり，機能要求のほかに，非機能要求（例外処理に関する要求を記述），外部インタフェース要求，単体テスト用のテストケース，および受入れテスト用のテストケースを明示した．なお，1C の受入れテスト用のテストケースは 18 件，2C は 11 件を用意した．これらは，PSP のプログラミング課題では明記されておらず，被験者の認識によってプログラム規模や開発時間がばらつくのを抑えるために課題を詳細化した．

3.5 適用プロセス

被験者には，課題 1C を実施するときには sPSP0 プロセスを，課題 2C を実施するときには sPSP2 プロセスを適用してもらった．これらのプロセスは，それぞれ PSP0 および PSP2 をもとにして，被験者の実際の開発作業に近いプロセスを定義したものである．例えば，PSP ではコーディング工程とコンパイル工程を別々に定義しているが，sPSP ではこれらを一つの工程として定義した．

sPSP2 プロセスに含まれる工程を次に示す．*印の付いた工程は，sPSP0 には含まれていない工程である．

- 計画立案
- 設計
- 設計レビュー *
- コーディング（コンパイルも含む）
- コードレビュー *
- 単体テスト
- 受入れテスト
- 事後分析

3.6 レビューの実施方法

CBR グループには，設計レビュー工程では表 1 のチェックリスト（11 項目）を，コードレビュー工程では表 2 のチェックリスト（24 項目）を用いて個人レビューを実施してもらった．TCBR グループには，図 1 に示したテストケース（11 件）および表 3 の PSP 欠陥型標準を与え，個人による設計レビューおよびコードレビューを実施してもらった．個人レビューは，設計の途中またはコーディングの途中ではなく，設計またはコーディング工程が完了してから実施した．

表 1　設計レビュー用のチェックリスト

分類	チェック項目
完全	・要求された機能は、設計で完全に網羅されている。
	・指定された出力は、全て作成されている。
	・必要な入力は、全て供給されている。
	・必要なファイル等の外部データは、全て把握している。
論理	・再帰呼び出し関数は、停止性が保証されている。
	・すべてのループは、初期設定され、増加され、正しく終了する。
データ構造	・必要なデータ構造は、全て定義されている。
	・それぞれのデータ構造の役割・機能は、明確になっている。
関数	・必要な関数は、全て列挙されている。
	・それぞれの関数の役割・機能は、明確になっている。
	・それぞれの関数のインタフェースは、明確に定義されている。

4. 実験結果

以下に，実験データの分析結果を示す．被験者は全部で 33 名であったが，時間内に課題を終えられなかった被験者や，プログラム規模が測定されていないなどデータに欠落のある被験者が合計で 10 名いたため，これらを除いた 23 名のデータについて分析した．

表 2　コードレビュー用のチェックリスト（一部）

分類	チェック項目
完全性	・設計内容は、コードで完全に網羅されている。
初期化	・プログラムの開始時に、変数とパラメータが初期化されている。
	・各ループの先頭で、変数とパラメータが初期化されている。
	・関数の先頭で、変数とパラメータが初期化されている。
関数呼び出し	・ポインタや＆を使用している場合、それらは正しく使用されている。
	・パラメータの数、データ型、順序は、正しく使用されている。
	・適切な戻り値が返されている。
演算	・=, ==, ‖ 等の演算子は、正しく使用されている。
	・暗黙の型変換など、演算の型は適切に使用されている。
	・不等号の向きや等号の有無は、正しく記述されている。
	・四則演算の優先順位や（ ）は、正しく使用されている。
制御	・すべてのループは、初期設定され、増加され、正しく終了する。
	・if else の構造は、正しく実現されている。

表 3　PSP 欠陥型標準（文献[3]より）

型番号	型名	型番号	型名
10	文書	60	チェック
20	構文	70	データ
30	ビルド、パッケージ	80	機能
40	割り当て	90	システム
50	インタフェース	100	環境

4.1 プロセス間のパフォーマンス比較

sPSP0 で課題 1C を実施したときの，規模，開発時間，生産性，およびプロセス品質データの分布を表 4 および図 2 に示す．プロセス品質は，単体テストで発見された KLOC 当り欠陥数，受入れテストで発見された KLOC 当り欠陥数，プロセス全体で作り込んだ KLOC 当り総欠陥数といった測定量で表している．

同様に，sPSP2 で課題 2C を実施したデータの分布を表 5 および図 3 に示す．課題 1C と比べると，プログラム規模が小さくなっているとともに，開発時間も短

くなっている．生産性に関しては全体的に低下しており，単体テスト欠陥および総欠陥についても課題 2C の方が低下している．その原因として，課題 2C はファイル入力処理を含んでおり，課題 1C とは異なるドメイン知識が被験者に求められたためと思われる．課題 2C では，被験者にとって不慣れなドメイン知識が要求されたために，生産性の低下や，欠陥を作り込みやすい状況になったものと考えられる．

表 4　被験者のパフォーマンス（sPSP0，課題 1C）

	規模 (LOC)	時間 (分)	生産性 (LOC/時)	単体ﾃｽﾄ 欠陥/KLOC	受入ﾃｽﾄ 欠陥/KLOC	欠陥 /KLOC
最大値	266	468	62.7	70.0	32.4	156.6
第三四分位点	175.3	341.0	36.9	33.3	10.3	61.2
中央値	144	293.5	30.2	17.1	6.9	47.6
第一四分位点	122	248.8	24.9	7.6	0.0	32.7
最小値	90	215	12.8	0.0	0.0	7.5
標準偏差	45.6	70.8	12.0	17.6	9.9	33.9

表 5　被験者のパフォーマンス（sPSP2，課題 2C）

	規模 (LOC)	時間 (分)	生産性 (LOC/時)	単体ﾃｽﾄ 欠陥/KLOC	受入ﾃｽﾄ 欠陥/KLOC	欠陥/KLOC
最大値	193	309	52.2	60.0	47.6	173.5
第三四分位点	113.0	217.5	38.6	27.4	2.6	98.4
中央値	88	204.0	24.5	22.0	0.0	87.9
第一四分位点	67	179.5	18.8	15.8	0.0	57.4
最小値	42	123	11.8	0.0	0.0	14.7
標準偏差	37.0	44.6	12.3	17.5	12.5	43.6

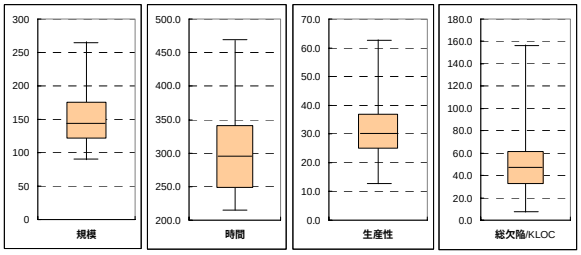

図 2　被験者のパフォーマンス（sPSP0，課題 1C）

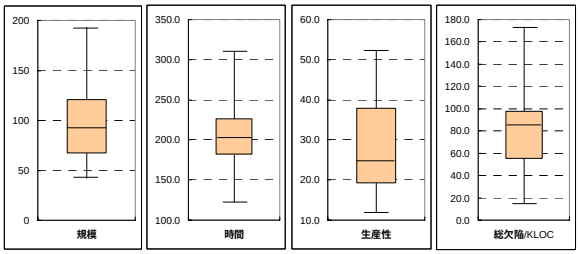

図 3　被験者のパフォーマンス（sPSP2，課題 2C）

しかしながら，表 4 と表 5 を比べると，受入れテストで発見された KLOC 当り欠陥数は，最大値を除けば課題 2C の方が少ない．受入れテストで発見される欠陥数は，受入れテスト用のテストケースの質によっても異なる．本実験の場合，同値分割技法により著者が作成した妥当なテストケースを被験者に与えている．したがって，テストケースの質に関する個人差はなく，また，課題間のテストケースの差よる影響はほとんどないと思われる．

被験者の多くは，課題 2C になってプログラム実現上の難易度が高まったために，プロセス全体（とくにコーディング工程）では多くの欠陥を作り込んでいたようである．しかし，受入れテスト工程に投入されたプログラム品質は課題 2C の方が相対的に良くなっているといえる．これは，sPSP2 により個人レビューを導入したことが品質向上に貢献していると考えられる．

4.2　レビュー手法の定量的比較

次に，sPSP2 のデータに関して，TCBR と CBR との比較を行う．表 6 に，主な測定量の比較結果を示す．表 6 のデータは，被験者数，総欠陥数，総開発時間が異なるため，欠陥数や工程時間を比較するには正規化する必要がある．ここでは，総欠陥数または総開発時間について正規化した値で比較する．

表 6　レビュー手法の比較

比較項目	CBR	TCBR
被験者数 (人)	13	10
作込み・除去欠陥総数 (個)	71	91
開発時間合計 (分)	2,604	2,124
設計作込み欠陥数 (個)	13 (18.3%)	6 (6.6%)
コーディング作込み欠陥数 (個)	51 (71.8%)	76 (83.5%)
設計レビュー除去欠陥数 (個)	5 (7.0%)	1 (1.1%)
コーディング除去欠陥数 (個)	30 (42.3%)	41 (45.1%)
ｺｰﾄﾞﾚﾋﾞｭｰ除去欠陥数 (個)	8 (11.3%)	18 (19.8%)
単体テスト除去欠陥数 (個)	24 (33.8%)	23 (25.3%)
設計レビュー時間合計 (分)	157 (6.0%)	79 (3.7%)
コードレビュー時間合計 (分)	297 (11.4%)	143 (6.7%)
ｺｰﾄﾞﾚﾋﾞｭｰ欠陥除去効率 (欠陥／時間)	1.82	7.55

※ カッコ内は総欠陥数または総開発時間との比率を表す

CBR および TCBR の両グループとも，欠陥の 70%以上はコーディング工程で作り込んでおり，設計工程で作り込んだ欠陥数と合計すると約 90%を占めている．一方，設計レビューおよびコードレビューで除去した欠陥の合計は，いずれのグループも約 20%であり，欠陥摘出率は決して高くはない．もっとも，コーディング工程で作り込んだ欠陥の多くはコーディング工程で除去しているため，欠陥摘出率が低い値になっていると考えられる．

なお，設計レビューで除去できた欠陥数が少ないため，設計レビューに関するレビュー手法の比較は適切ではない．したがって，以降ではコードレビューに関

するレビュー手法の比較を中心に議論する．

コードレビューの除去欠陥数に着目すると，TCBR の除去欠陥比率が 19.8%であるのに対して，CBR の比率は 11.3%と低い値である．また，コードレビュー欠陥除去効率，すなわち，レビュー時間当りの除去した欠陥数について，TCBR は 1 時間当り 7.55 個の欠陥を除去しているが, CBR では 1 時間当り 1.82 個にとどまっている．これらの結果から，今回の実験データについて，コードレビューに関しては CBR よりも TCBR の方が効果的かつ効率的に欠陥除去を行うことができたといえる．

また，設計レビュー時間およびコードレビュー時間の総開発時間に対する比率を比較すると，CBR が合計で 17.4%であるのに対して，TCBR は合計で 10.4%と低い．また，設計レビューおよびコードレビューの工程別時間を一人当り時間に換算して比較しても，TCBR の方が短い時間でレビューを行っている．レビュー時間は，チェックリスト項目数とテストケース項目数の影響を受けるため一概には結論づけられないが，今回の実験に関しては，TCBR の方が効率的にレビューを実施できていたと考えられる．その要因として，TCBR では設計レビューとコードレビューで同一のテストケースを使用しており，被験者がテストケースの内容に慣れたことが影響しているものと考えられる．

4.3 欠陥型別の比較

表 7 に，CBR と TCBR のそれぞれについて，コードレビューにより除去された欠陥数を欠陥型別に示す．除去欠陥数が少ないため，一般的な結論づけはできないものの，TCBR の方がインタフェースに関する欠陥を多く除去できていることがわかる．これは，テストケースに基づいてプログラムの処理フローを追跡するため，モジュール間インタフェースの欠陥を発見しやすかったためと考えられる．

表 7　欠陥型別のコードレビュー除去欠陥数

欠陥型	CBR	TCBR
10: 文書	0	1
20: 構文	4	6
40: 割り当て	1	1
50: インタフェース	0	4
60: チェック	0	2
70: データ	1	0
80: 機能	2	4
合計	8	18

また，機能に関する欠陥も，TCBR は CBR に比べて多くの欠陥を摘出できている．これは，一般的なテストケースは主に機能テストを行うためのものであるため，機能に関する欠陥が多く摘出できているものと考えられる．

5. 考察

5.1 TCBR 手法の有効性

今回の実験では，TCBR 手法をコードレビューに適用した際の効率性および有効性を示すことができた．効率性が発揮される要因として，チェックリストの項目数よりもテストケースの項目数が少ないことと，設計レビューとコードレビューで同一のテストケースを利用するために被験者のテストケースに対する理解度や慣れが高まっていたことが挙げられる．また，有効性に関して，プログラムの処理フローを追跡することによって，関数モジュール間のインタフェースや，処理ルーチンの機能的な欠陥を発見しやすかったことが要因として考えられる．

一方で，例えばファイルのクローズし忘れなど，TCBR よりも CBR の方が検出しやすい欠陥もある．本稿の段階では，このような TCBR により発見が容易でない欠陥に関する分析がまだ行えていない．このような欠陥に関する分析を行うとともに，TCBR において発見するための工夫や，または TCBR と CBR の組合せによるレビュー手法などの検討が求められる．

また，今回の実験ではコードレビューに TCBR を適用したが，この場合，単体テストを実施した方が効率的である可能性がある．今回は，検出欠陥数が少なかったために設計レビューで発見された欠陥について分析が行えなかったが，コードレビューよりも設計レビューの方が TCBR の効率性および有効性が発揮されるものと予想される．

5.2 実験方法による影響

今回の実験は，新人研修のなかの演習を利用して実施したため，プログラミング課題の内容や規模などにある程度の制約が生じた．プログラミング課題の規模が大きくなればテスト項目数も増加し，TCBR によるレビュー時間も増大する．課題の規模が大きくなったときに，依然として TCBR の方が CBR よりも有効であるのかどうか評価が求められる．

また，経験豊富な被験者もいたものの，プログラミング経験の浅い被験者も少なからず存在していた．被験者の経験レベルによってレビューで検出される欠陥

は異なるため，TCBR の実適用における有効性を評価するためには,経験者を対象とした評価が求められる．

5.3 TCBR 手法の実適用

今回の実験では，受入れテスト用のテストケースをあらかじめ作成し，これを被験者に与えてレビューを実施してもらった．しかし，本手法を実際に適用する際には，テストケースを前もって作成する工数を含めた全体的な効率を評価する必要がある．

実際には，テストケース数は膨大になるため，縮約したテストケースを選択し，優先順位付けを行って実施する技法が求められる．Thelin らの研究では，ユースケースにユーザ視点による優先順位付けをするために AHP を用いている[5]．しかし，AHP による優先順位付けは，理論的には優れているものの，比較対象が多くなると一対比較の回数が増大するため現実的ではない．直交表を用いてテストケースを縮約するなどの工夫が求められる．

5.4 本実験の副次的効果

本実験は研修を利用して実施したこともあり，被験者には演習に関するレポートを提出してもらった．その結果，個人レビューに対して否定的な見解も若干あったものの，レビューの重要性を認識するよい機会になったという感想が多数を占めた．文献[11]にあるように，実務の場面において体系的なレビュー手法が適用されていないだけでなく，レビューすら実施されていない場合が多い．このような演習を通じて，ソフトウェア技術者にレビューの重要性を伝達できたことは,実験本来の目的とは異なるがひとつの成果といえる．

6. おわりに

本稿では，個人のソフトウェア開発作業における設計成果物およびソースコードを対象に，TCBR と CBR という 2 つのレビュー手法の比較実験を行った．研修という場における実験という制約や，得られたデータ数が十分ではないため結論は限定的であるものの，TCBR の有効性および効率性，ならびに摘出しやすい欠陥型の傾向を示すことができた．

今後の課題として，評価すべき要因を抽出しての制御実験を行うことで，TCBR の効果および適用範囲を明らかにする必要がある．また，テストケース作成の工数を含めた全体的な効率性の検討または評価や，プログラミング課題の規模を大きくした場合の評価などが求められる．

参考文献

[1] Wiegers, K.E.: *Peer Reviews in Software – A Practical Guide*, Addison-Wesley, 2002 (渡辺洋子訳: ピアレビュー，日経 BP 社, 2003).

[2] Gilb, T. and Graham, D.: *Software Inspection*, Addison-Wesley, 1993 (富野壽監訳: ソフトウェア・インスペクション，共立出版, 2000).

[3] Humphrey, W.S.: *A Discipline for Software Engineering*, Addison-Wesley, 1995 (松本正雄監訳: パーソナルソフトウェアプロセス技法，共立出版, 1997).

[4] Lanubile, F., Mallardo, T., Calefato, F., Denger, C. and Ciolkowski, M.: “Assessing the Impact of Active Guideline for Defect Detection: A Replicated Experiment,” *Proc. of 10th Intl. Symposium on Softw. Metrics*, IEEE Computer Society, 2004, pp. 269-278.

[5] Thelin, T., Andersson, C., Runeson, P. and Dzamashvili-Fogelström, N.: “A Replicated Experiment of Usage-Based and Checklist-Based Reading,” *Proc. of 10th Intl. Symposium on Softw. Metrics*, IEEE Computer Society, 2004, pp. 246-256.

[6] Thelin, T., Runeson, P. and Wholin, C.: “An Experimental Comparison of Usage-Based and Checklist-Based Reading,” *IEEE Trans. on SoftwEng.*, Vol. 29, No. 8, 2003, pp. 687-704.

[7] Thelin, T., Runeson, P. and Wohlin C.: “Prioritized Use Cases as a Vehicle for Software Inspections,” *IEEE Software*, Vol. 20, No. 4, 2003, pp. 30-33.

[8] 松川文一, Sabaliauskaite, G., 楠本真二, 井上克郎: UML で記述された設計仕様書を対象としたレビュー手法 CBR と PBR の比較評価実験，オブジェクト指向最前線 2002，近代科学社，2002，pp. 67-74.

[9] Beck, K.: *eXtreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.

[10] JUnit.org, http://www.junit.org.

[11] Ciolkowski, M., Laitenberger, O. and Biffl, S.: “Software Reviews: The State of the Practice,” *IEEE Software*, Vol. 20, No.6, 2003, pp. 45-51.

[12] Basili, V.R., Shull, F. and Lanubile, F.: “Building Knowledge through Families of Experiments,” *IEEE Trans. on SoftwEng.*, Vol. 25, No. 4, 1999, pp. 456-473.

[13] Shull, F., Rus, I. and Basili, V.: “How Perspective--Based Reading Can Improve Requirements Inspections,” *IEEE Computer*, Vol. 33, No. 7, July 2000, pp. 73–79.

[14] Porter, A., Votta, L.G. and Basili, V.R.: “Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment,” *IEEE Trans. on SoftwEng.*, Vol. 21, No. 6, 1995, pp. 563-575.