

論文

過去の不具合の修正工数を考慮したソフトウェアレビュー手法

森崎 修司[†] 久保 匡^{††} 荻野 利彦^{††} 阪本 太志^{††}
山田 淳^{†††}

An Approach for Software Inspection by Considering Correction Efforts of Defects in Past Development

Shuji MORISAKI[†], Tadashi KUBO^{††}, Toshihiko OGINO^{††}, Takashi SAKAMOTO^{††},
and Atsushi YAMADA^{†††}

あらまし 同一ソフトウェアの過去のバージョンや類似のソフトウェアで検出された不具合の修正工数の情報を用いて、修正工数低減の効果が大きい欠陥をレビューで検出する手法を提案する。提案手法では、レビューの準備段階においてレビュー対象のソフトウェアの過去のバージョンや類似のソフトウェアのテスト工程において検出された不具合のうち、修正工数が大きくなっている欠陥種別を特定する。欠陥検出の際に、その欠陥種別を優先して検出すべき欠陥種別（観点）として個々のレビューアに割り当てる。レビューアは観点到該当する欠陥の検出を試みる。商用開発において作成された設計ドキュメントと過去のバージョンのテストの際に収集された不具合の修正工数の情報を用いてケーススタディを実施し、修正工数の情報から欠陥種別を特定できるかどうか、レビュー観点として割り当てることによりレビューで見逃すと修正工数が大きくなるような欠陥が検出できるかどうかを確認した。過去の不具合の修正工数の情報から観点を設定でき、検出した欠陥をレビューで見逃していた場合に必要になったと予想される修正工数の合計が、提案手法を用いない場合と比較して、大きくなる結果が得られた。

キーワード 観点設定, ソフトウェアインスペクション

1. ま え が き

ソフトウェアレビューはプログラムが動作する前の段階で、ソフトウェアに含まれる欠陥を目視により検出するソフトウェアの品質向上活動である。ソフトウェアレビューの効果は、欠陥をテストで発見、修正するよりも早い段階で発見、修正することにより、再テストの工数、及び、誤ったドキュメントに沿って開発を進める工数を低減することにより、もたらされる。例えば、欠陥が含まれる設計ドキュメントをもとにコーディングが実施され、その欠陥がテストで検出された場合、誤りのある設計ドキュメント自体の修正に加え、

ソースコードの修正、修正部分のテスト、回帰テストが必要になるからである。

従来のレビュー手法の多くは、どのような欠陥であつても早期発見により修正コストの低減が可能であることを前提とし、網羅的な欠陥の検出を目的としている。しかしながら、網羅的に欠陥検出ができたとしても、修正コスト低減の効果が小さい軽微な欠陥の割合が増えると欠陥修正コスト低減の効果が得られにくくなる。例えば、ドキュメントの体裁、誤字脱字の欠陥はテストでの修正工数と要求、設計、コーディング時で検出された場合の修正工数が大きく変わらない欠陥といえる。可読性の観点で、このような欠陥の検出は必要であるが、その割合が大きくなるとソフトウェアレビューによる早期発見、早期修正の効果が得られにくくなる。

本研究では、欠陥の早期発見、及び、早期発見による修正工数の低減を目的とし、過去の不具合の修正工数の情報から見逃すと修正工数が大きくなる欠陥を検出するための観点を設定し、欠陥の検出を試み

[†] 静岡大学情報学部, 浜松市

Faculty of Informatics, Shizuoka University, 3-5-1 Johoku, Naka, Hamamatsu-shi, 432-8011 Japan

^{††} 東芝デジタルメディアエンジニアリング株式会社, 青梅市

TOSHIBA Digital Media Engineering Corporation, Ome-shi, 198-0024 Japan

^{†††} (株) 東芝ソフトウェア技術センター, 川崎市

Corporate Software Engineering Center, TOSHIBA Corporation, Kawasaki-shi, 212-8583 Japan

るレビュー手法を提案する．具体的には，レビューの準備段階で既に開発が完了した類似ソフトウェアで開発中に発見された不具合の修正工数の傾向を分析し，レビュー観点を挙げる．レビューの実施において観点をレビューアに割り当て，個々のレビューアは観点に沿った欠陥の検出を試みる．商用開発で開発されたソフトウェアを対象として実施したケーススタディにおいて，過去の不具合の修正工数の情報から観点が設定できるかどうか，また，観点を設定することにより修正工数低減の効果が高い欠陥が検出できるかどうかを試行する．

以降，2. で提案手法を，3. では商用開発プロジェクトにおけるケーススタディを示し，4. で考察する．5. で関連研究を述べ，6. でまとめる．

2. 提案手法

2.1 概要

提案手法は四つの手順からなる．手順を実施する作業者はアナリスト，モデレータ，レビューアである．アナリストは，レビュー対象のソフトウェアや関連するプロジェクトの知識をもち，不具合の修正工数の情報（不具合管理表の情報）をもとに，どのような観点に着目すれば，欠陥検出が可能かを検討し，観点候補を列挙する．モデレータは，レビュー対象，レビューアのスキル，知識を把握し，アナリストが列挙した観点候補をレビューアに割り当てる．レビューを実施す

る際にモデレータはレビューの司会進行を務める．レビューアは割り当てられた観点で欠陥検出を試みる．アナリスト，モデレータは同一の作業者が担当してもよい．なお，本論文では，レビューで検出された欠陥とテストで検出された欠陥を区別するために，テストで検出された欠陥を不具合と呼ぶ．本論文中の欠陥は，全てレビューで検出された欠陥を指す．

提案手法の概要を図1に示す．具体的には次の手順で実施する．

(1) 不具合種別の抽出

アナリストが過去の不具合管理表 D から修正工数の大きな不具合種別を抽出する．

(2) 観点候補の列挙

アナリストは抽出した不具合種別からレビューで検出を注力する観点の候補を列挙する．

(3) 観点の割当

モデレータは観点候補，レビューアのスキル，対象ソフトウェア/プロジェクトに求められる品質や利用できるコストを勘案し，観点候補をレビューアへ割り当てる．

(4) 欠陥検出

それぞれのレビューアは自身に割り当てられた観点に沿った欠陥を検出する．

手順1で用いる不具合管理表 D は表1に示すような形式のものである．表1の各行は1件の不具合に対応する．項目には修正工数 e が必ず含まれるものとす

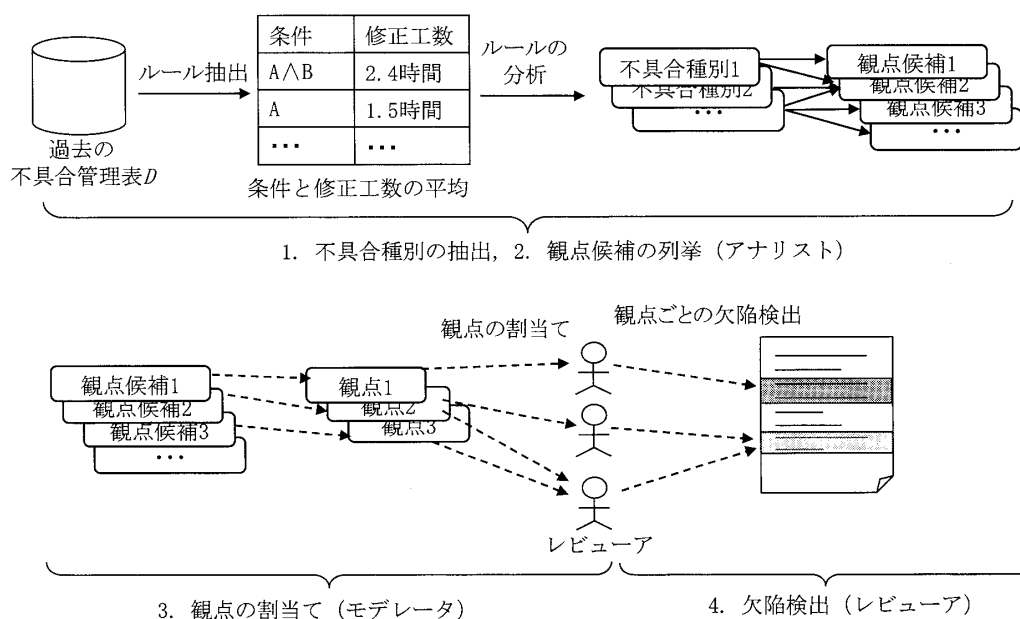


図1 提案手法の概要

Fig. 1 Overview of proposed approach.

表 1 対象とする不具合管理表 $\mathcal{D}$
Table 1 Bug tracking information $\mathcal{D}$.

不具合 ID	項目 l_1^c	...	項目 l_m^c	項目 l_{m+1}^f	...	項目 l_{m+u}^f	修正工数 e
1	d_{11}^c	...	d_{m1}^c	$d_{(m+1)1}^f$	...	$d_{(m+u)1}^f$	e_1
2	d_{12}^c	...	d_{m2}^c	$d_{(m+1)2}^f$	...	$d_{(m+u)2}^f$	e_2
...							
n	d_{1n}^c	...	d_{mn}^c	$d_{(m+1)n}^f$	...	$d_{(m+u)n}^f$	e_n

る．他の列は不具合 ID, 報告者, 機能名, 発生原因, 混入原因, 混入工程, テスト種別等の名義尺度, 順序尺度の項目 (項目名 $l_k^c (1 \leq k \leq m)$) と不具合の症状, 原因, 対応結果等の自由記述式の項目, 及び, 発見日時, 修正日時等の間隔尺度, 比尺度の項目 (項目名 $l_k^f (m < k \leq m+u)$) から構成される. k 番目の項目 (項目名 $l_k^c (1 \leq k \leq m)$) がとり得る値の集合 C_k を $C_k = \{c_{ki} | (1 \leq i \leq n_k)\}$, 不具合 ID が $j (1 \leq j \leq n)$ である不具合を D_j , 不具合 D_j の修正工数を e_j , D_j の項目 l_k^c の値を $d_{kj}^c (d_{kj}^c \in C_k)$ とする.

例えば, 3 段階の重要度カテゴリをもつ “重要度” の項目は次のようになる. $l_k = \text{重要度}, C_k = \{\text{高}, \text{中}, \text{低}\}$.

2.2 実施手順

2.2.1 不具合種類の抽出

アナリストは $A_t \Rightarrow \text{effort}(a_t)$ のようなルールの形式で, 不具合の条件と条件を満たす不具合の修正工数の平均を抽出する. 具体的には, 不具合管理表 $\mathcal{D}$ から $l_k^c (1 \leq k \leq m)$ と修正工数 $e_j (1 \leq j \leq n)$ から与えられた出現頻度 f_{min} 件以上出現する条件の組合せ, 条件を満たす不具合の修正工数の平均, を全て求める. ここで,

$$\begin{aligned} A_t &= A_{t1} \wedge A_{t2} \wedge \dots \wedge A_{ts_t} \\ &= (l_{p_1}^c, v_{t1}) \wedge (l_{p_2}^c, v_{t2}) \wedge \dots \wedge (l_{p_{s_t}}^c, v_{ts_t}) \end{aligned}$$

であり, $1 \leq p_1 < \dots < p_{s_t} \leq m$, また, (l, v) は項目名 l の値が v であることを表す. 例えば, 重要度が高い不具合 (項目 “重要度” が値 “高”) の場合, (重要度, 高) と表す.

なお, 間隔尺度, 比尺度の d^c は順序尺度に変換してからルール抽出する.

$\mathcal{D}$ のうち $(l_{p_1}^c, v_{t1}) \wedge (l_{p_2}^c, v_{t2}) \wedge \dots \wedge (l_{p_{s_t}}^c, v_{ts_t})$ を満たす不具合の修正工数の平均値 a_t , 及び, A_t の $\mathcal{D}$ での出現頻度 f_t を次から得る.

// ルールの構成要素が最大 max_length, $\mathcal{D}$ での出現頻度が f_{min} 以上のルール frequent_rules[1, ..., max_length] を抽出する.

```
// 構成要素が 1 つ ( $A_{t1}$ ) のルールの抽出 ( $A_{t1} \Rightarrow \text{effort}(a_t)$ )
frequent_rules[1] = null;
for k = 1 to m do
  for i = 1 to  $n_k$  do
    A = ( $l_k^c, c_{ki}$ );
    effort_values[] =  $\mathcal{D}.\text{get\_effort\_values}(A)$ ;
    if (|effort_values[]| >=  $f_{min}$ ) then
      frequent_rules[1].add(A, mean(effort_values[]));
    end if
  end for
end for
```

```
// 構成要素が 2 つ以上のルールの抽出
// ( $A_{t1} \wedge A_{t2} \dots \Rightarrow \text{effort}(a_t)$ )
for j = 2 to max_length do
  frequent_rules[j] = null;
  for A in frequent_rules[j - 1] do
    for i = 1 to  $n_k$  do
      A' = A;
      A'.add( $l_k^c, c_{ki}$ );
      effort_values[] =  $\mathcal{D}.\text{get\_effort\_values}(A')$ ;
      if (|effort_values[]| >=  $f_{min}$ ) then
        frequent_rules[j].add(A', mean(effort_values[]));
      end if
    end for
  end for
end for
```

図 2 ルール抽出アルゴリズム
Fig. 2 Rule Extraction Algorithm.

$$a_t = \frac{\sum_{j=1}^n e_j}{f_t}, f_t = |\{D_j\}|$$

ただし, j は次を全て満たす.

$$d_{p_1 j}^c = v_{t1}, d_{p_2 j}^c = v_{t2}, \dots, d_{p_{s_t} j}^c = v_{ts_t}$$

具体的には図 2 に示すアルゴリズムを用いて, 最低出現頻度 f_{min} とルールの最大構成要素数 (A_{ts_t} の s_t の最大数 (図中 max_length) をパラメータとして与え, 該当する全てのルール (図中 frequent_rules) を得る.

提案手法では, f_{min} を n の 1% 程度に設定してルールを抽出し, 次節の手順により, 修正工数の大きなルールや出現頻度の大きいルールのうちからレビューでの検出により, 修正工数の低減に有効と思われるルールを目視によって選ぶ. なお, 不具合管理表からのルール抽出の詳細は文献 [4] でも報告している.

2.2.2 観点候補の列挙

アナリストは得られたルールの修正工数の平均値 a_t , ルールの出現頻度 f_t をもとにルールを選出し, レビュー観点の候補 $F = \{F_1, F_2, \dots, F_n\}$ を得る. 具体的には, 修正工数が大きいルール (a_t の値が上位のルール) を選び, 修正に多くの工数を要する不具合種別を明らかにする. また, a_t の値はそれほど小さくなくても出現頻度 f_t が大きなルールがあれば, 修正工数が大きいルールとして不具合種別を明らかにする.

次に, アナリストは, 絞り込んだルールが示す条件 (d^c), 及び, 条件を満たす不具合の集合の自由記述 (d^f) から, 観点候補を挙げる. 観点は, どのような着眼点で欠陥を検出しようとするれば, それらの欠陥が見逃されずに検出できていたかを端的に示したものである. ルールを構成する A_{ti} から観点候補が直接設定できる場合と A_t を構成する各項目 A_{ti} をもとにアナリストが検討の上, 設定する場合がある. 例えば, 修正工数が大きくなる l_p として “報告者”, その値 d_p^c として “報告者 A” があった場合, これをそのまま F に加えても欠陥の検出が容易であるとは限らない. “報告者 A が検出している不具合種別” という観点だけでは, レビューアがレビュー対象から欠陥を検出できない可能性があるからである. そこで, d_{pk}^c が “報告者 A” となっている自由記述 (d_{pk}^f) の内容をアナリストが目視し, その共通点や検出を導く観点を F に加える. 例えば, 報告者 A が担当するテストの多くが性能評価に関するものであり, 発見された性能問題を修正するために修正工数が多く必要とされていると判断されれば, それを F に加える.

2.2.3 観点の割当

レビュー観点の候補 F をレビュー対象やレビューアの欠陥検出スキルに合わせてモデレータが割り当てる. レビューア r_k の観点の集合を F_{r_k} としたとき, $F_{r_k} = \{f_{r_k1}, \dots, f_{r_k n_{r_k}}\}$ ただし, $f_{r_k i} \in F$, $1 \leq n_{r_k} \leq |F|$. 同一の観点を複数のレビューアに割り当ててもよく, 必ずしも $F_{r_i} \cap F_{r_j} = \phi$ となる必要はない. 一般には, 経験の多いレビューアには性能問題等, 相対的に検出が難しい観点や広範囲を調べなければ検出できない観点を割り振る. 文献 [3] で報告されている perspective-based reading の perspective の割当と同様に, スキルが十分でないレビューアが担当できるものから観点を想定し, スキルが十分にあるレビューアにはスキルが十分でないレビューアが担当できない観点を割り当てる.

2.2.4 欠陥検出

レビューア r_k は, 自身に割り当てられた観点 F_{r_k} をもとにレビュー対象に含まれる欠陥の検出を試みる. 検出には, 設定した観点による目視部分の局所化, 目視で確認する順番等をレビューアが勘案しながら, レビュー対象を読み進めていく.

3. ケーススタディ

3.1 対象

ケーススタディとして, 東芝デジタルメディアエンジニアリングで開発したソフトウェアに提案手法を適用し, 試行した. 対象ソフトウェアの概要を表 2 に示す. 対象ソフトウェアはサブシステムごとに異なるプロジェクトで開発されている. 各サブシステムでは, 少なくとも, 一つ以上の旧バージョンの開発プロジェクトが完了しており, その不具合管理表がある. 対象とした不具合管理表の収集項目を表 3 (試行 1), 表 4 (試行 2) に示す. 試行 1, 2 は異なる表 2 のソフトウェアの異なるサブシステムであり, 異なるチームで開発している. 二つのプロジェクト間では不具合管理表の項

表 2 対象プロジェクトと観点抽出の対象とした不具合管理表
Table 2 Overview of the project and the bug tracking information in the case study.

概要	デジタルメディア処理ソフトウェア (デジタルテレビの機能を模したメディア処理を行う, 組込み応用ソフトウェア開発)
開発期間	3.5 年
ソースコード規模	約 650,000 行 (C/C++)
開発プロセス	ウォーターフォール (派生開発)
観点抽出に利用した不具合管理表の収集フェーズ	単体試験～総合試験
観点抽出に利用した不具合管理表の不具合件数	350 件 (不具合以外は含まれない)

論文／過去の不具合の修正工数を考慮したソフトウェアレビュー手法

表 3 試行 1 で対象とした不具合管理表の収集項目
Table 3 Bug tracking information in trial 1.

項目名	値
修正工数	修正に要した工数
担当者	テスト担当者名
サブシステムカテゴリ	5 種類
サブシステム名	53 種類 (機能名等)
原因種別	設計ミス, 設計漏れ, 調査漏れ, 調査ミス, 外部設計漏れ, 外部設計ミス, 内部設計漏れ, 内部設計ミス, コーディング漏れ, コーディングミス
検査種別	正常系, 異常系, 限界系, その他
発生種別	新規不具合, 内在不具合, 既知不具合, その他

表 4 試行 2 で対象とした不具合管理表の収集項目
Table 4 Bug tracking information in trial 2.

項目名	値
修正工数	修正に要した工数
不具合ランク	A (重度), B (中度), C (軽度)
担当者グループ	A～G までのグループ
機種 ID	A, B, C
サブシステム名	53 種類 (機能名等)
原因種別	設計ミス, 設計漏れ, 仕様変更, 委託先不具合, 周知不足, 未実装, コーディング漏れ, コーディングミス
二次原因分類	設計検証不足, 潜在的な問題, 動作検証不足
修正箇所	インタフェース処理, エラー処理, その他, 基本処理, 限界処理, その他
発見すべき工程	単体検査, 結合検査, 総合評価
品質を作り込むべき工程	仕様検討, 設計, コーディング
検査種別	正常系, 異常系, 限界系, その他
発生種別	新規不具合, 内在不具合, 既知不具合, その他
要因	内部要因, 外部要因

目 1 は異なる。

対象プロジェクトでは、不具合の発見日時や修正日時等の日時の情報を収集している。日時の情報は間隔尺度であるため、順序尺度に変換することにより、提案手法で用いるルール抽出の対象データとできるが、区間の設定をはじめ適切な変換方法の決定は本研究の次のステップに該当すると考え、本ケーススタディでは対象としていない。なお、修正工数以外の比尺度の収集項目は存在しなかった。

3.2 方 法

提案手法を適用したレビューを試行 1-P, 2-P とする。それぞれのプロジェクトで収集された過去のバージョンの不具合管理表から、観点の候補が挙げられるか試行する。観点の候補を挙げることができれば、既に完了しているレビュー（試行 1-C, 2-C）の結果と、検出できた欠陥をもしも見逃した場合の予想修正工数を比較する。試行 1-P, 2-P と試行 1-C, 2-C をそれぞれ比較し、試行 1-P, 2-P のほうが見逃した場合の予想修正工数が大きくなれば、提案手法による観点設定の効果が得られたものとする。

試行 1-P と 1-C, 及び 2-P と 2-C に参加したレビューアは異なる。試行 1-P, 2-P に参加したレビュー

アも、類似サブシステムの開発に携わっているメンバから選出し、対象ソフトウェアやプロジェクトに十分な知識があるレビューアとする。レビューアの人数は試行 1-P が 3 名、試行 1-C が 5 名、試行 2-P, 2-C はともに 3 名である。

試行 1-C, 2-C のレビューでは、レビューアに明示的な観点の割当は行われないが、長期にわたって開発を共にしているレビューアどうしで欠陥検出に関して役割分担が十分にできている。具体的には、固有のアルゴリズムが正しく実現できていない、起こり得る例外の対処漏れや対処誤りといった対象ソフトウェア自体の欠陥の検出、OS, ミドルウェア, ハードウェア等の要素技術との組合せにおいて発生し得る欠陥の検出、複数のモジュールにまたがって複合的に起こる性能欠陥の検出、特殊なユーザ利用環境において起こる欠陥の検出、が含まれ、個々のレビューアがこれらの観点における自身の役割分担（専門領域としての分担）を意識している。そのため、異なるレビューアが同一の観点から欠陥を検出することはほとんどない。

ケーススタディでは、最大構成要素数 s_t を 3 とし、 f_{min} として 5 を与え、2.2.1 で示したアルゴリズムを実装したプロトタイププログラムを用いてルールを抽

出する．それらのルールのうち， a_t が上位のもの 50 件程度， f_t が上位のもの 50 件程度を目視で確認する．

3.3 結果

3.3.1 観点抽出

試行 1-P, 2-P において，それぞれ 3000 件程度のルールを抽出した．抽出したルールのうち， a_t , f_t が上位のものをそれぞれを 50 件程度目視し，観点を設定した．目視と観点設定に要した時間は試行 1-P, 2-P とも 30 分程度である．試行 1-P, 2-P で得られたルールを，表 5, 表 6 に示す．また，それらルールをもとに試行で選択したレビュー観点を表 7 に示す．

表 5 の R1-1 に該当する不具合から，これらをレビューで検出するための観点を検討した結果，内部処理の順番を検討する内部シーケンス，デジタルメディア処理のためのアルゴリズム，に着目することで，レビューにおいてこれらの欠陥の検出が期待できると判断し，観点として設定した．同様に R1-2, R1-4 から，例外処理や特殊入力データ/パラメータ，パラメータ初期化/クリアを観点として設定した．R1-3 から，ユースケース，排他処理に着目することで，レビューにおいてこれらの欠陥の検出が期待できると判断し，

観点として設定した．

表 6 の R2-1 に該当する不具合から，これらをレビューで検出するための観点を検討した結果，基本処理の処理順序，状態遷移中，リソース競合時の処理順序に着目することで，レビューにおいてこれらの欠陥の検出が期待できると判断し，観点として設定した．同様に R2-2 から例外処理，ユースケース，基本処理の処理順序を観点として設定した．R2-3 からは，基本処理の処理順序，パラメータ設定を観点として設定した．

3.3.2 欠陥検出

全ての試行での検出件数と検出した欠陥を見逃した場合の予想修正工数を表 8 に示す．表中の欠陥数は誤字脱字等の軽微な欠陥を除いている．試行 1-P, 2-P は，提案手法による観点設定をした上でレビューを実施したものである．試行 1-C, 2-C は対象ソフトウェア開発時に実施した通常の観点（修正工数を勘案したシステマチックな手順によらず設定した観点）でレビューを実施したものである．表 8 中の試行 1-C, 2-C のカッコ付きの数値は，試行 1-P, 2-P で設定した観点に該当する欠陥の数を示している．試行 1-P と 1-C を

表 5 試行 1-P で不具合管理表から抽出されたルール（抜粋）

Table 5 Selected rules extracted from the bug tracking information in trial 1-P.

ID	ルール	出現頻度	平均修正時間
R1-1	(サブシステムカテゴリ=MW(P)) ∧ (原因種別=内部設計ミス)	3.8%	6.62
R1-2	(担当者=015) ∧ (検査種別=異常系)	3.5%	6.33
R1-3	(原因種別=内部設計漏れ)	10.2%	5.68
R1-4	(サブシステムカテゴリ=MW(F)) ∧ (検査種別=異常系)	4.4%	5.67

表 6 試行 2-P で不具合管理表から抽出されたルール（抜粋）

Table 6 Rules extracted from the bug tracking information in trial 2-P.

ID	ルール	出現頻度	平均修正時間
R2-1	(機種=C) ∧ (原因分類=設計ミス) ∧ (修正箇所=基本処理)	13.0%	10.5
R2-2	(機種=C) ∧ (原因種別=設計漏れ) ∧ (修正箇所=基本処理)	6.9%	12.9
R2-3	(担当グループ=F) ∧ (品質を作り込むべき工程=設計) ∧ (修正箇所=基本処理)	16.8%	13.6

表 7. 試行で選択したレビュー観点

Table 7 Focused defect types in the trials.

試行 1-P	試行 2-P
例外処理	例外処理
パラメータ初期化/クリア	状態遷移中，リソース競合時の処理順序
特殊入力データ/パラメータ	ユースケース
排他処理	基本処理の処理順序
内部シーケンス	パラメータ設定
ユースケース	
アルゴリズム	

表 8 試行の結果
Table 8 Result of the trials.

	試行 1-P	試行 1-C	試行 2-P	試行 2-C
提案手法による観点設定	有	無	有	無
レビュー工数（人時）	3.0	6.0	7.7	9.0
レビューアの人数	3	5	3	3
検出欠陥の合計	8 件	6 件	11 件	4 件
観点に該当する欠陥	6 件	(2 件)	10 件	(3 件)
見逃した場合の予想修正工数の合計（人日）	15.5	6.5	14.4	5.0
観点に該当する欠陥の予想修正工数の合計（人日）	13.0	(2.0)	14.0	(5.0)
観点に該当しない欠陥の予想修正工数の合計（人日）	2.5	4.5	0.4	0.0

表 9 観点による検出欠陥の分類-試行 1
Table 9 Defect types in trial 1.

分類	試行 1-P で設定した観点に該当するか	試行 1-P		試行 1-C	
		件数	予想修正工数	件数	予想修正工数
例外処理	該当する	1	3	1	1
内部シーケンス	該当する	3	7	1	1
パラメータ初期化/クリア	該当する	1	1.5	0	-
排他処理	該当する	1	1.5	0	-
特殊入力データ/パラメータ	該当する	0	-	0	-
ユースケース	該当する	0	-	0	-
アルゴリズム	該当する	0	-	0	-
不明瞭な表現	該当しない	1	0.5	1	0.5
機能仕様	該当しない	1	2	0	-
インタフェース仕様	該当しない	0	-	3	4

表 10 観点による検出欠陥の分類-試行 2
Table 10 Defect types in trial 2.

分類	試行 2-P で設定した観点に該当するか	試行 2-P		試行 2-C	
		件数	予想修正工数	件数	予想修正工数
例外処理	該当する	3	6.9	1	2
基本処理の処理順序	該当する	1	0.6	2	3
パラメータ設定値	該当する	2	2.6	0	-
ユースケース（の網羅性）	該当する	2	1.3	0	-
パラメータの初期化タイミング	該当する	1	0.6	0	-
あらゆるタイミングにおける処理順序	該当する	1	2	0	-
インタフェース仕様	該当しない	1	0.4	0	-

比較するとレビュー工数に 2 倍の差があるが、主にレビューアの人数の差による影響であり、試行 1-P, 2-P は試行 1-C, 2-C と比較してレビュー実施時間に大きな差はなかった。また、全ての試行において、提案手法によるレビューの方が検出件数が大きくなり、見逃した場合の予想修正工数が大きくなった。提案手法によって観点を設定した試行 1-P, 2-P では、検出された欠陥の 7 割以上が設定した観点に該当するものであった。提案手法によって観点を設定しなかった試行 1-C, 2-C においても、試行 1-P, 2-P で設定した観点に該当する欠陥が検出されていた。

表 9 は、試行 1-P, 1-C で検出された欠陥を分類したものである。観点として設定した 7 種類の欠陥分類のうち、提案手法による観点設定をした試行 1-P では 4 種類、6 件の欠陥が検出された。これらの欠陥を見

逃した場合の予想修正工数の合計は 13 人日であった。また、設定した観点に該当しない 2 種類の欠陥（不明瞭な表現、機能仕様）もそれぞれ 1 件ずつ検出された。提案手法を適用しなかった試行 1-C のレビューでも、試行 1-P で設定した 2 種類の観点（例外処理、内部シーケンス）に該当する欠陥がそれぞれ 1 件検出された。これらを見逃した場合の予想修正工数の合計は 2.0 人日であった。試行 1-C ではそれ以外に 2 種類の欠陥種別（不明瞭な表現、インタフェース仕様）に関する欠陥が合計 4 件検出された。これらを見逃した場合の予想修正工数の合計は 4.5 人日であった。

表 10 は、試行 2-P, 2-C で検出された欠陥を分類したものである。試行 2-P では、設定された全ての観点から 1 件以上の欠陥が検出された。これらを見逃した場合の予想修正工数の合計は 14 人日であった。試

行 2-P では、設定した観点に該当しない欠陥は 1 種類（インタフェース仕様）で 1 件のみであった。試行 2-C でも試行 2-P で設定した 2 種類の観点（例外処理、基本処理の処理順序）に該当する欠陥 3 件（それぞれ、1 件、2 件）が検出された。それらを見逃した場合の予想修正工数の合計は 5 人日であった。

4. 考 察

4.1 観点候補の列挙

過去のバージョンの開発において収集された不具合管理表の情報を元にしてレビュー観点の候補を設定することができた。アナリストとの議論の中で、今回設定した観点の多くは修正コストに影響を与えるものであり、過去のバージョンの不具合情報なしに同じ観点を設定することは難しいものであったという意見を得た。

ルールの構成要素 A_{ti} は不具合管理表の項目 I_k ととり得る値 C_k に限定されるため、必ずしも A_{ti} から直接、観点を設定することはできなかったが、アナリストのプロジェクトの知識やルールを満たす不具合の個々の記述 I^f を目視し、共通点を検討することにより、観点を設定できた。例えば、試行 1-P の R1-2 (表 5) の“担当者=015”はアナリストが担当者 015 が実施したテスト項目を把握しており、その項目と担当者 015 が記録した不具合の自由記述部分を勘案しながら、観点“特殊入力データ/パラメータ”を設定できた。不具合管理表の項目 I^c の定義により I^c の項目から直接観点が得られる検討すれば、プロジェクトに関する知識をアナリストが十分にもたない場合でも、観点の列挙が可能になることが期待される。

4.2 観点設定による欠陥検出

試行 1-P, 2-P では、設定した観点に該当する欠陥が多く検出された。検出された全ての欠陥に対する観点に該当する欠陥の割合は、75.0% (試行 1-P), 90.1% (試行 2-P) であり、検出された欠陥全てが観点に該当するものではなかったが、多くの欠陥が設定した観点に該当するものであった。検出した欠陥がレビューで見逃され、テストで修正することになった場合の予想修正工数の合計は提案手法を用いた試行 1-P, 2-P の方が大きな値となった。観点割当てによる効果が得られたものと推測される。軽微な欠陥を除く欠陥の件数も試行 1-P, 2-P の方が大きくなっている。試行 1-C, 2-C においても暗黙的な役割分担による観点設定ができていたため、やみくもに観点を設定するだけで件数が増えたわけではなく、4.1 で述べた過去の不具合情

報に基づいて適切な観点を設定できたことによるものと考えられる。

試行 1-P, 2-P のモデレータ、レビューアから次のような意見が得られた。

- 普段よりも普段よりも欠陥件数（軽微なもののぞく）が多くなったように感じた。
- レビューに慣れていないレビューアには「こういう点を重点的にみてください」と提示するのは効果があるだろう。
- 経験が多い人にとっては、観点に沿った欠陥をしようとすると、かえって制約になる可能性がある。
- 観点だけ与えられても想像ができない場合があるので補足説明が必要である。

提案手法の方がレビュー工数が小さかった。試行 1-1 と試行 1-2 の差はレビューアの人数によるところが大きく、提案手法のレビュー工数が大きく減少したわけではない。この点についてレビューアに意見を求めたところ、観点によっては確認すべきドキュメントの箇所が減少するものがあつたり、終了判断が短時間でできたりしたが、特にレビュー工数が小さくなったという実感はなかった、という意見が得られた。

4.3 妥 当 性

それぞれの試行は異なるレビューアで実施しているため、検出数や検出した欠陥の予想修正工数を単純に比較し結論づけることは難しい。しかしながら、予想修正工数、検出数をはじめとする収集データ、レビューアの意見の両方から、提案手法により見逃した場合の修正工数が大きな欠陥を検出できたり、深刻な欠陥の検出数が大きくなったりする結果が得られた。

ケーススタディで収集したレビューで見逃した場合の予測修正工数の合計は、レビューアの見積りによるものである。テストで発見、修正する際に、修正箇所の影響範囲やデグレードを防ぐための修正確認テストの規模を想定してもらう等して、なるべく客観的な方法での見積りとなるようにしてもらったが、誤差を含む可能性がある。

ルールからレビュー観点を設定するためにはアナリストの知識が必要になる。アナリストへのインタビューからは、プロジェクトの知識やテストや開発担当者の作業が把握できていれば、観点を設定することは可能である、という意見を得た。不具合管理表の混入原因の項目にレビュー観点として設定しやすい選択肢を設けることにより、ルールからレビュー観点の設定が容易になることが期待される。

5. 関連研究

これまで研究されているレビュー手法においても、リーディングテクニックと呼ぶ読み進め方によって、本研究と同様に検出すべき欠陥を明確にしようとしている。Defect based Reading [6] は、レビューアごとに異なるシナリオ（読み進め手順）を割り当て、レビューで検出すべき欠陥を明確にしている。文献[6]では、データの一貫性の欠如、誤った機能定義、欠落している機能の三つのシナリオで実験が実施され、その効果が確認されている。DBR でも原理的には本研究と同様に優先して検出する欠陥を設定できるが、修正工数を加味したシナリオの割当てに関する言及はない。また、シナリオの作成においても修正工数を考慮することは明示されていない。

Usage-based reading [7] は、ソフトウェアの利用者の視点から欠陥を検出する。具体的には、利用シナリオを挙げ、利用者にとって重要な順番に利用シナリオを順位付けし、重要な利用シナリオから順に利用シナリオにそって欠陥の検出を試みる。シナリオには優先順位付けに修正工数を加味することは原理的には可能であるが、言及されていない。Time controlled reading [5] も usage-based reading と同様に利用シナリオの重要度に応じて欠陥検出にかかる時間を割り当てて、修正工数による時間の割当には言及がない。

文献[2]において Chernak は、チェックリストに含まれる個々の質問項目の優先順位を付けるために、質問項目ごとに検出できた欠陥の数を計数している。Chernak の研究でも欠陥の修正工数を考慮することは原理的には可能であると言及はない。

6. む す び

修正工数低減の効果が大きい欠陥をレビューで検出することを目的とし、以前のバージョンや類似のソフトウェアで検出された不具合の修正工数の情報を用いて、レビュー観点を設定し、欠陥を検出する手法を提案した。提案手法で用いる不具合の情報はレビュー対象の過去のバージョンや類似のソフトウェアのテスト工程において検出された不具合を記録した不具合管理表である。不具合管理表から修正工数に影響を与えている混入原因や機能名等を勘案し、過去のバージョンや類似のソフトウェアにおいて修正工数が大きかった不具合種別を調査し、レビュー観点の候補として挙げる。レビューにおいては、観点の候補を個々のレビュー

アに割り当てる。レビューアは観点に該当する欠陥の検出を試みる。

提案手法を評価するため、商用開発において作成されたソフトウェア設計ドキュメントと以前のバージョンのテストの際に用いられた不具合管理表を用いて、ケーススタディを実施した。具体的には、不具合管理表の情報からレビュー観点の候補が挙げることができかどうか、挙げた候補をレビュー観点としてレビューアに割り当てることによりレビューで見逃すと修正工数が大きくなるような欠陥が検出できるかどうかを二つの設計ドキュメントを用いて試行した。ケーススタディの結果、過去のバージョンの不具合管理表から観点候補を挙げることができた。また、観点候補をレビューアに割り当てることにより、提案手法を用いない場合と比較して、レビューで見逃した場合の修正工数の合計が大きくなった。また、軽微な欠陥を除く欠陥件数が増えた。提案手法における観点の候補の列举や観点に沿ったレビューは大きなコストを必要としないため、レビューの効率を高めることができる。

謝辞 試行に協力頂いた方々に深く感謝する。本研究は株式会社東芝ソフトウェア技術センターとの共同研究の一環として、東芝デジタルメディアエンジニアリング株式会社と実施した。本研究の一部は、文部科学省「次世代 IT 基盤構築のための研究開発」の委託、同省科学研究援助費（若手 B: 課題番号 21700033, 基盤研究 (B): 23300009）の助成を受けた。

文 献

- [1] V. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, and M. Zelkowitz, "The empirical investigation of perspective-based reading," J. Empirical Software Engineering, vol.2, no.1, pp.133-164, 2006.
- [2] Y. Chernak, "A statistical approach to the inspection checklist formal synthesis and improvement," IEEE Trans. Softw. Eng., vol.22, no.12, pp.866-874, 1996.
- [3] C. Denger and F. Shull, "A practical approach for quality-driven inspections," IEEE Softw., vol.24, no.2, pp.79-86, 2007.
- [4] S. Morisaki, A. Monden, T. Matsumura, H. Tamada, and K. Matsumoto, "Defect data analysis based on extended association rule mining," Proc. International Workshop on Mining Software Repository, pp.17-24, 2007.
- [5] K. Petersen, K. Ronkko, and C. Wohlin, "The impact of time controlled reading on software inspection effectiveness and efficiency," Proc. International Symposium on Empirical Software Engineering and Measurement, pp.139-148, 2008.

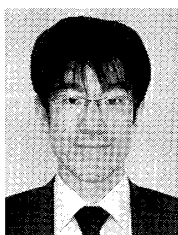
- [6] A. Porter, L. Votta, and V. Basili, "Comparing detection methods for software requirements inspections: A replicated experiment," *IEEE Trans. Softw. Eng.*, vol.21, no.6, pp.563-575, 1995.
- [7] T. Thelin, P. Runeson, and B. Regnell, "Usage-based reading? An experiment to guide reviewers with use cases," *J. Information and Software Technology*, vol.43, no.15, pp.925-938, 2001.

(平成 23 年 12 月 21 日受付, 24 年 4 月 6 日再受付)



山田 淳 (正員)

1984 大阪大学大学院工学研究科電気工学修士課程了。同年 (株) 東芝入社。現在, 東芝ソフトウェア技術センタープロセス・品質技術開発担当主幹。東芝グループ内のソフトウェアプロセス改善推進活動に従事。情報処理学会, IEEE 各会員, 情報処理学会規格調査会 ISO/IEC-JTC1/SC7 委員。



森崎 修司 (正員)

2001 奈良先端科学技術大学院大学情報科学研究科博士後期課程了。同年 (株) インターネットイニシアティブ入社。2012 静岡大学情報学部講師。博士 (工学)。ソフトウェアレビュー, エンベリカルソフトウェア工学の研究に従事。情報処理学会, プロジェクトマネジメント学会, IEEE 各会員。



久保 匡

東芝デジタルメディアエンジニアリング (株) ソフトウェアセンター AV プラットフォームソフトウェア担当シニアエンジニア。デジタルテレビの開発に従事。



荻野 利彦

東芝デジタルメディアエンジニアリング (株) ソフトウェアセンター AV プラットフォームソフトウェア担当シニアエンジニア。デジタルテレビの開発に従事。



阪本 太志

東芝デジタルメディアエンジニアリング (株) ソフトウェアセンターシニアエンジニア Company SEPG に従事。IPA/SEC プロセス改善 WG 委員, 日科技連ソフトウェア品質管理研究会主査, JASPIC 副運営委員長。