特別研究報告

題目

シナリオを用いたレビュー手法 PBR の追証実験
- UML で記述された設計仕様書を対象として -

指導教官
井上 克郎 教授

報告者
松川 文一

平成 14 年 2 月 19 日

大阪大学 基礎工学部 情報科学科

平成 13 年度 特別研究報告

シナリオを用いたレビュー手法 PBR の追証実験
- UML で記述された設計仕様書を対象として -

松川 文一

内容梗概

ソフトウェアレビューの目的は, ソフトウェアの開発過程で生成されるプロダクトを検査し, バグを検出, 除去することにある. 開発の初期の段階でバグを取り除いておくことによって, その後の開発段階でのコストを抑え, ソフトウェアの品質と生産性がより高いものとなる.

現在までに数多くのレビュー手法が提案されている．最近では，Perspective-based Reading(PBR) 手法が注目されている．Laitenberger らは，オブジェクト指向開発で標準的に用いられている仕様記述言語 UnifiedModeling Language(UML) で記述された設計仕様書を対象として，PBR を，従来手法の一つである Checklist-based Reading(CBR) 手法との間で有効性の比較実験を行った．しかし，この実験で用いられた UML 図は 2 種類であり, より様々な UML 図に対して比較実験を行う必要性が指摘されている.

本研究では, 5 種類の UML 図を用いた設計仕様書に対し, CBR と PBR の有効性を比較するための追証実験を実施した．実験では情報科学科の 3 年生 59 人を被験者とし, レビュー対象として 2 つのシステムの UML 図を用いた. 被験者はあらかじめ複数のバグを埋め込んだ UML 図を，指定されたレビュー手法でチェックし, 検出したバグ, 検出時刻等を報告した. 収集データについて，バグの検出率，検出時間に関して，PBR を使用した被験者と CBR を使用した被験者の間で比較を行った．結果として，構文的なバグの検出に関しては CBR に, 意味的なバグ，他の図と関連したバグの検出に関しては PBR に有効性が認められた.

主な用語

ソフトウェアレビュー (Software Review)
UML(Unified Modeling Language)
PBR(Perspective-based Reading)
CBR(Checklist-based Reading)

目 次

1 まえがき **4**

2 準備 **6**

2.1 ソフトウェアレビュー 6

2.1.1 プロセス 7

2.1.2 プロダクト 8

2.1.3 役割 8

2.1.4 手法 8

2.2 Checklist-based Reading 9

2.3 Perspective-based Reading 9

2.4 Unified Modeling Language 10

3 CBR と PBR の比較評価実験 **16**

3.1 概要 16

3.2 実験詳細 16

3.2.1 レビュー対象システム 16

3.2.2 バグ 17

3.2.3 人数配分 21

3.2.4 実験データ 21

4 実験データの分析と考察 **22**

4.1 個人データ分析 22

4.1.1 手法による違い 22

4.1.2 バグの種類別 22

4.1.3 UML 図別 22

4.2 チームデータ分析 24

4.2.1 チームの組み方 24

4.2.2 グループ化 24

4.2.3 データ算出 24

4.2.4 データ比較 25

4.3 考察 26

5 まとめと今後の課題 **28**

謝辞 **29**

参考文献 **30**

付録 **32**

1　まえがき

ソフトウェアの品質を向上させるのに有効な手段の 1 つとして, ソフトウェアレビューがある. ソフトウェアレビューとはソフトウェアの開発工程において作成されたプロダクト (設計ドキュメント, プログラムコードなど) の内容について, プロダクト作成者と他の開発者の間で議論し, 最終的にプロダクト内にバグが残らないようにする作業のことである. ソフトウェアのプロダクトに対するレビューによって開発プロセスの初期の段階でバグを除去すれば, ソフトウェアの開発コストを削減することができ, その結果としてソフトウェアの品質と生産性を向上させることができる.

ソフトウェアレビューは一般的にいくつかの作業 (計画, バグ検出, 会議 (バグ収集), バグ修正等) から成る. そのうちの 1 つであるバグ検出において, これまでにさまざまな手法が提案されている. それまでのバグ検出は, レビュー者に対して特に何の指針も無く, プロダクトのみで行っていた. それゆえ, レビュー者は自身の経験や直感に頼らざるを得なかった. そこで **Checklist-based Reading(CBR)** が提案された. これはプロダクトと共にいくつかのチェック項目が記されたチェックリストを用いて, そのチェック項目に yes/no 形式でチェックしながらレビューを進めていくという手法であり, バグ検出の効率を上げることができる. さらに最近では, 1997 年に Basili によって Scenario-based Reading(SBR) が提案されている. この手法は, シナリオと呼ばれる, レビューの進め方や着目すべき情報等が記されたものを用いている. SBR にはさらにさまざまな形式があり, その中のひとつである **Perspective-based Reading(PBR)** は, プロダクトに関わるいくつかの立場による視点からレビューを行うという手法である.

また近年, オブジェクト指向ソフトウェア開発が主流となり, その開発に **UML(Unified Modeling Language)** が用いられていることが多い. UML は Booch らによって開発されたソフトウェア設計開発言語であり, さまざまな種類の図を用いてソフトウェアの体系を視覚的に表現することができる.

Leitenberger らはこの UML を用いた設計仕様書に対し CBR と PBR を用いてレビュー実験を行い, 両手法の比較評価を行った. この実験では, PBR は CBR に対し, バグの検出において 58%, バグ検出に費やすコストにおいては 41%の有効性を示したという結果が報告されている [9]. しかしこの実験において用いられた UML 図は 2 種類 (クラス図, コラボレーション図) であった. UML は現在数多くの種類の図が存在し, それらに対しても両手法の比較評価を行う必要があると指摘されている.

そこで本研究では, プロダクトに, さらに多くの種類の UML 図を用いて CBR, PBR の両手法でレビュー実験を大阪大学の学生に対して行い, 追証実験という形で両手法の比較評価を行うことを目的とする.

以降, 2 節では UML, ソフトウェアレビュー及び CBR, PBR の両手法について説明する. 3 節では今回行ったレビュー実験について述べる. 4 節ではレビュー実験によって得られたデータに対し分析及び考察を行う. 最後に 5 節では, まとめと今後の課題について述べる.

2　準備

2.1　ソフトウェアレビュー

IEEE 標準 [5] によると,「ソフトウェアレビューとは, 開発計画から想定される状況からのズレ (矛盾) を突き止め, その解消のために改善を促すことを目的として, ソフトウェアの構成要素やプロジェクトの現状を評価する活動である. 」と定義されている. ソフトウェアの構成要素とはプロジェクト計画書, 要求仕様書, 設計仕様書, テスト労力書, ソースコード等, 開発途中で作成されるすべてのものである. これらのプロダクトについて, 作成者と他の開発者の間で議論し, 最終的にプロダクト内にバグが残らないようにすることにより, 開発プロセスの初期の段階でバグを除去すれば, ソフトウェアの開発コストを削減することができ, 結果としてソフトウェアの品質と生産性を向上させることができる.

通常, ソフトウェアレビューは 2 つのタイプに分類される. 管理レビューと技術レビューである. 管理レビューでは, レビューチームがプロジェクトレベルの計画とそれに対するプロジェクトの現状を評価する. 一方, 技術レビューでは, 設計ドキュメントからテストまでのあらゆる生成物の評価を行う.

本研究では, 設計工程で生成される設計ドキュメントに対して行われる設計レビューに着目する.

技術的な観点から [10], ソフトウェアのレビューには図 1 に示す 4 つの項目について決定する必要がある. 各項目については以下で説明する.

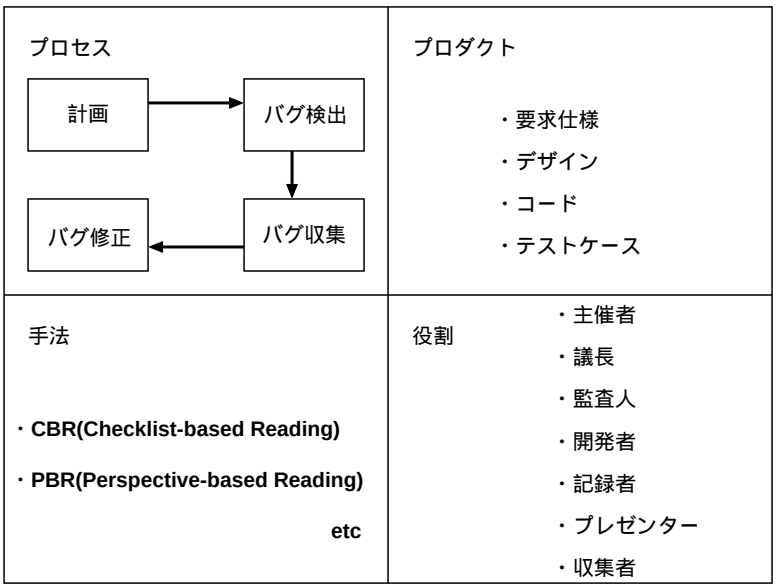


図 1: ソフトウェアレビューの技術的側面

2.1.1 プロセス

レビューのプロセスの各ステージを図 2 に示す. レビューのプロセスは主に 4 ステージから成る.

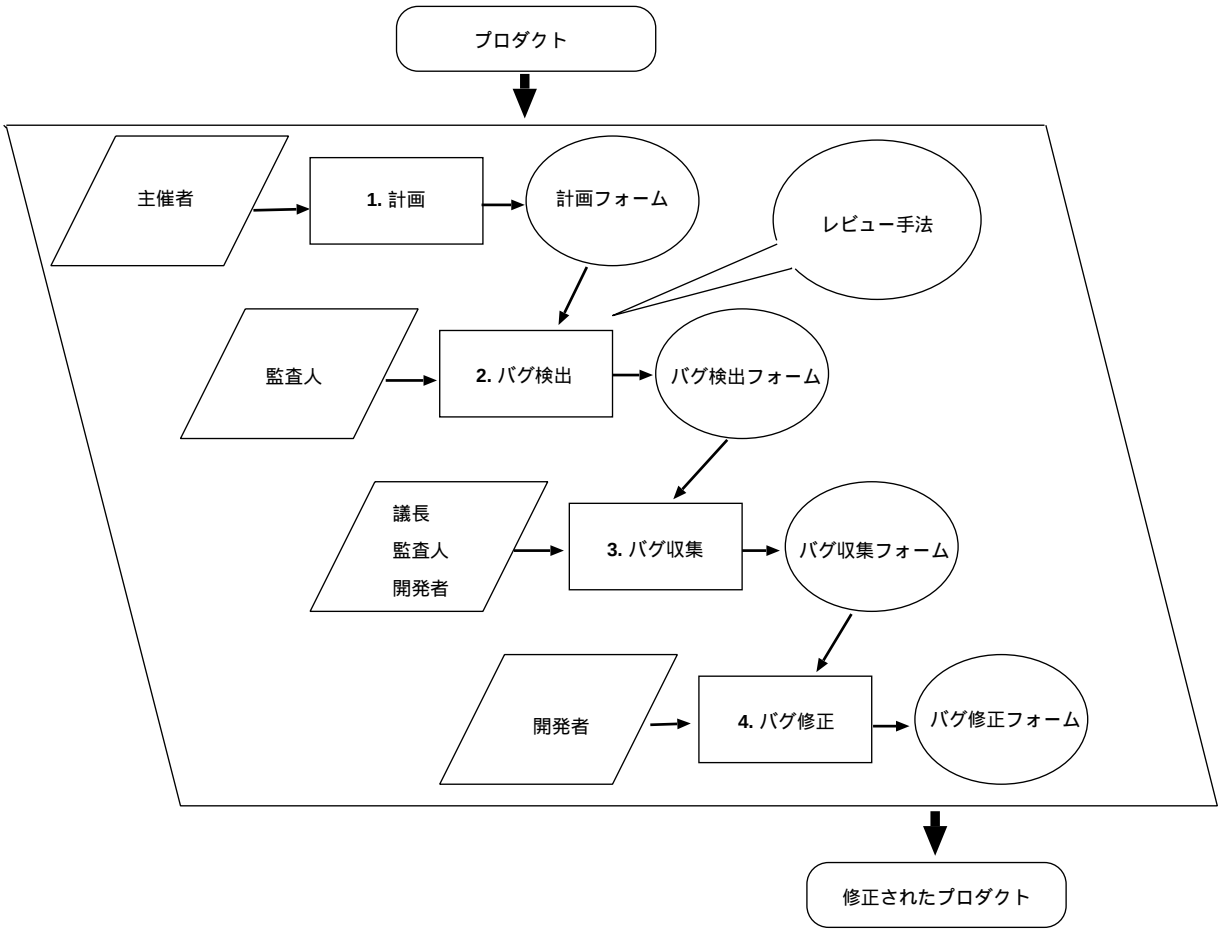


図 2: ソフトウェアレビューのプロセス

1. 計画
   参加者の招集, 各役割の割当て, スケジューリング, マテリアルの分配等. これは主催者によって行われる.

2. バグ検出
   監査人がレビュー手法を用いて, 分配されたプロダクトを検査し, バグを検出する.

3. バグ収集
   個別に検出されたバグをグループ会議で収集, 議論し, 記録する. 監査人, 議長, そして

開発者が参加することが多い.

4. バグ修正
   開発者が, 検出されたバグを除去し, 改めて開発を行う.

2.1.2 プロダクト

レビュー対象となるプロダクトの種類としては, 要求仕様, デザイン, ソースコード, テストケースがある. その中でもレビューは大抵, 要求仕様やコードモジュールなど, テキストドキュメントに対して行われることが多い. また, テストのみに比べ, コードでのレビューにより 39%, デザインのレビューにより 44%, バグによるコストが抑えられたという結果も確認されている [10].

2.1.3 役割

レビューにおける各役割について以下に示す.

- 主催者 - レビュー全体の計画を立てる.
- 議長 - レビューの各プロセスをまとめる.
- 監査人 - 対象となるプロダクトに対してバグの検出を行う.
- 開発者 - レビュー対象となるプロダクトを開発し, また修正後のプロダクトを開発する.
- 記録者 - レビュー会議で, 検出されたバグを記録する.
- 収集者 - 監査人によって検出されたバグをまとめる.

2.1.4 手法

バグの検出手法には, これまでにさまざまな手法が提案されている.

- Checklist-based Reading - 質問が並べられたリストを元にレビューを行う
- Scenario-based Reading - レビューの実施方法, 順序等を記したシナリオを元にレビューを行う. これにはさらにさまざまな種類がある.
  - Defect-based Reading - レビュー者がそれぞれ異なったバグに着目している
  - Function Point based - シナリオが特定のファンクションポイント (入出力, ファイル等) に着目した質問で構成されている

– Perspective-based Reading - さまざまな立場 (ユーザ, 設計者, テスタ等) を想定し, それぞれの視点でレビューを行う

本研究ではこの中の Checklist-based Reading, Perspective-based Reading の 2 つをとり挙げる.

2.2 Checklist-based Reading

Checklist-based Reading(CBR) は, レビュー対象プロダクトと共に, いくつかのチェック項目が記されたリストを使用し, そのチェック項目に yes/no 形式で答えながらレビューを行う手法である.
それまでのレビュー手法は, レビュー実施者に特に何の指針も無くただプロダクトのみが渡されるという形であった. レビュー効率を上げ, バグ検出に費やすコストを抑えることを目的として CBR が開発され, 用いられるようになった.

CBR のチェックリストは基本的に以下の様に構成される.[9]

- *"Where to look"* - 可能性のある問題点のリスト
- *"How to look"* - 各問題点についてバグを特定する指針となる

CBR を用いると, 質問に yes/no で答えながら進めていくので, レビューは容易に行うことができる反面, プロダクトの全体の情報を網羅する必要があるため, 1 回のレビューに費やす時間や負担は大きくなってしまうという欠点がある.

2.3 Perspective-based Reading

Perspective-based Reading(PBR) は, レビューの実行方法等を記したシナリオを用いたレビュー手法 Scenario-based Reading(SBR) のさまざまな種類の内のひとつである.
対象となるソフトウェアに関わる者のいくつかの異なった視点 (ユーザ, 設計者, テスタ等) から, それぞれのシナリオを用いてプロダクトをレビューし, バグを検出する.

PBR のシナリオは主に 3 つの部分から成る.

- *Introduction* - 対象プロダクトの要求品質を記述した部分
- *Instruction* - どのドキュメントを使用するか等, レビュー全体の指針となる部分
- *Questions* - レビュー者がレビュー中に答える質問の部分

PBR では各視点ごとでレビュー対象となるドキュメント数が異なっているので, レビューにかかる時間や負担を抑えることができる.

本研究における実験では, ユーザ, 設計者, プログラマという 3 つの視点を設定し, レビューを行った. 図 3, 図 4 にそれぞれチェックリスト, シナリオの例 (一部) を示す.

2.4 Unified Modeling Language

UML(Unified Modeling Language) とは, G. Booch, J. Rumbaugh, I. Jacobson によって考案されたソフトウェア開発言語である [15]. クライアント, 分析者, そして開発者間の誤解や間違いを解消するために, 全員が共通の理解のもとに使用できる標準的な表記法の必要性から, この言語が開発された.

UML は, 表 1 に示すダイアグラムから成る.

表 1: UML のダイアグラム

種類	ダイアグラム名	説明
	ユースケース図	システムの動作をユーザの視点で示したもの
静的構造	クラス図	各クラス (クラス名, 属性, 操作を持つ) の関連を示したもの
	オブジェクト図	クラスの特定のインスタンスを示したもの
振る舞い	ステートチャート図	各クラスの状態の遷移を示したもの
	アクティビティ図	ユースケースやオブジェクトの振る舞いの流れを示したもの
	シーケンス図	オブジェクト間の相互作用を時系列に沿って示したもの
	コラボレーション図	オブジェクトの相互関係を示したもの
実装関係	コンポーネント図	システムを構成する物理的な部分 (ソフトウェアコンポーネント) を表現したもの
	配置図	システムの物理的な構成 (アーキテクチャ) を表現したもの

これらを用いることで, 開発に関わるメンバーのプロジェクトに対する理解を深め, 設計から開発までのプロセスを整合的に理解することができる.

本研究では, ユースケース図, クラス図, ステートチャート図, シーケンス図, コンポーネント図を扱う. 各図の例について図 5 から図 9 に示す.

チェックリスト

クラス図，ステートチャート図，シーケンス図，コンポーネント図に対して，以下の項目についてチェックしてください．チェックした際にバグを見つけた時には， 図上のバグの箇所に印をつけて，バグ調査表に記入してください．

クラス図			
1.	クラス図と要求仕様書の間に矛盾はありませんか？	☐ yes	☐ no
2.	必要なクラス，関連が全て定義されていますか？	☐ yes	☐ no
3.	冗長なクラスはありませんか？	☐ yes	☐ no
4.	全ての関連について多重度は定義されていますか？	☐ yes	☐ no
5.	プログラムを書く上で十分な情報が記入されていますか？	☐ yes	☐ no
ステートチャート図			
6.	ステートチャート図と要求仕様書の間に矛盾はありませんか？	☐ yes	☐ no
7.	必要な状態と遷移が全て定義されていますか？	☐ yes	☐ no
8.	状態の順番は正しいですか？	☐ yes	☐ no
9.	冗長な状態や遷移はありませんか？	☐ yes	☐ no
シーケンス図			
10.	シーケンス図，クラス図，要求仕様書の間で文法的な矛盾がありませんか?	☐ yes	☐ no

図 3: チェックリスト

チェックリスト（プログラマ）

あなたは，このシステムのプログラマであると思ってください．プログラミングをする上で，要求仕様書とユースケース図から作成されたクラス図，シーケンス図，コンポーネント図が正しくできていることが重要です．そこで，プログラマの立場で，クラス図とコンポーネント図にバグがないかどうかをチェックしてもらいます．
チェックは，以下の Step1～Step3 に従って行ってください．各 Step では指定された設計図を用意して，チェック項目に従って確認してください．バグを発見した時には，図上のバグの箇所に印をつけて，バグ調査表に記入してください．

Step 1	クラス図，ユースケース図，要求仕様書をチェックします．	
	ユースケース図上の各ユースケースの横に，そのユースケースを実行するのに必要なオブジェクトクラス名を書いて，以下の質問に答えなさい．	
	1.1.	ユースケースを実行する上で，必要な全てのオブジェクトクラスがクラス図上に定義されていますか？
	1.2.	ユースケースを実行する上で，クラス図上クラス間に必要な関連がありますか？
	1.3.	全ての関連について多重性が定義されていますか？
	1.4.	クラス図は，あなたがソースコードを作成するとした時に，十分な情報を含んでいると思いますか？
Step 2	コンポーネント図とクラス図をチェックする	
	コンポーネント図はシステムのソフトウェアコンポーネント間の関係を示しています．各コンポーネントは複数のオブジェクトクラスから構成されます．各コンポーネントに対して，コンポーネントを実装する上で必要と思われるクラスをリストアップして，以下の質問に答えなさい．	
	2.1.	クラス図の各クラスは，少なくとも一つのコンポーネントに含まれていますか？

図 4: シナリオ

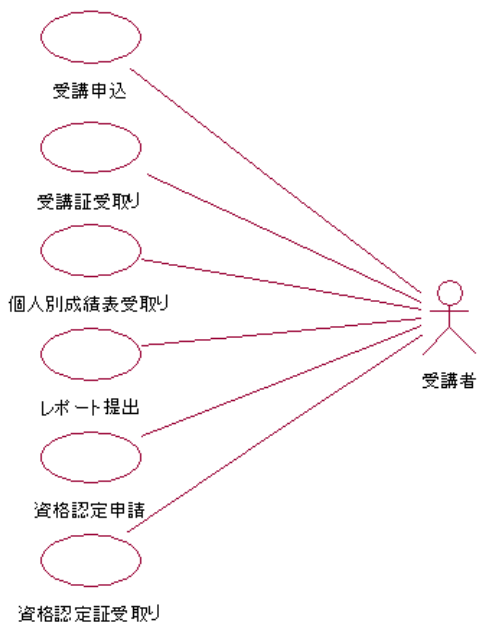


図 5: ユースケース図

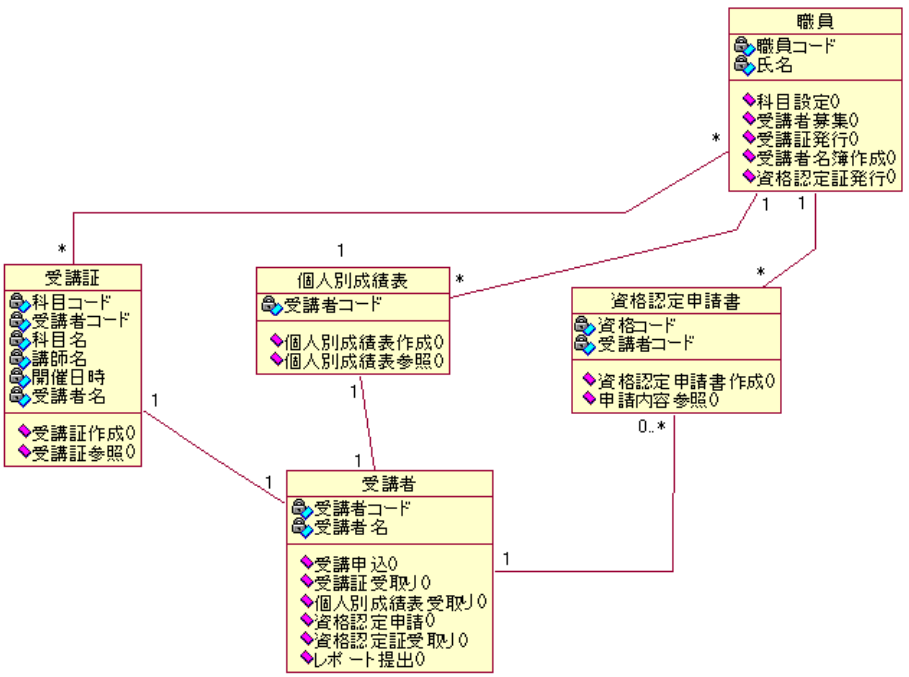


図 6: クラス図

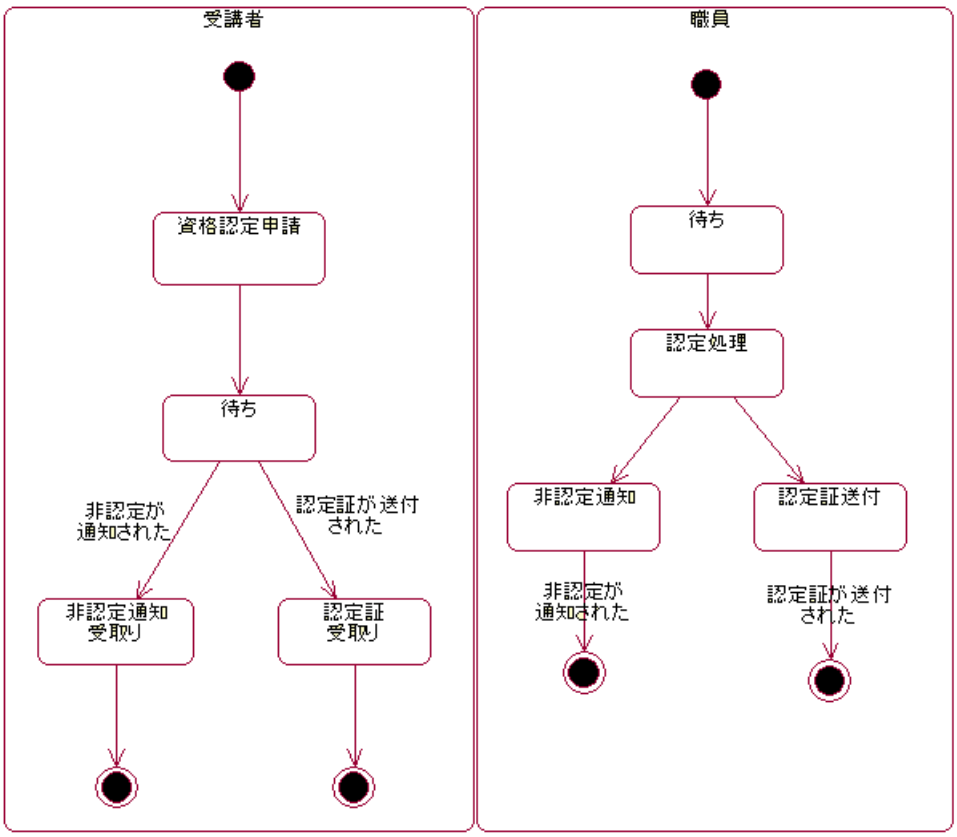


図 7: ステートチャート図

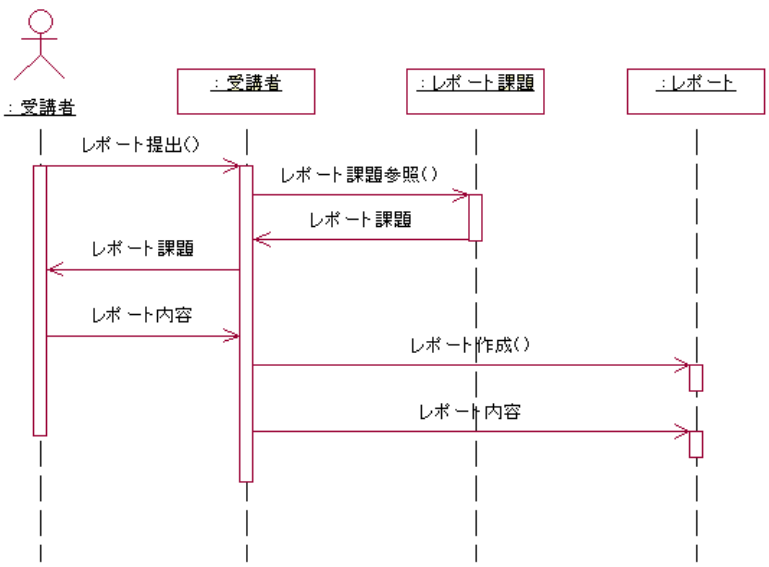
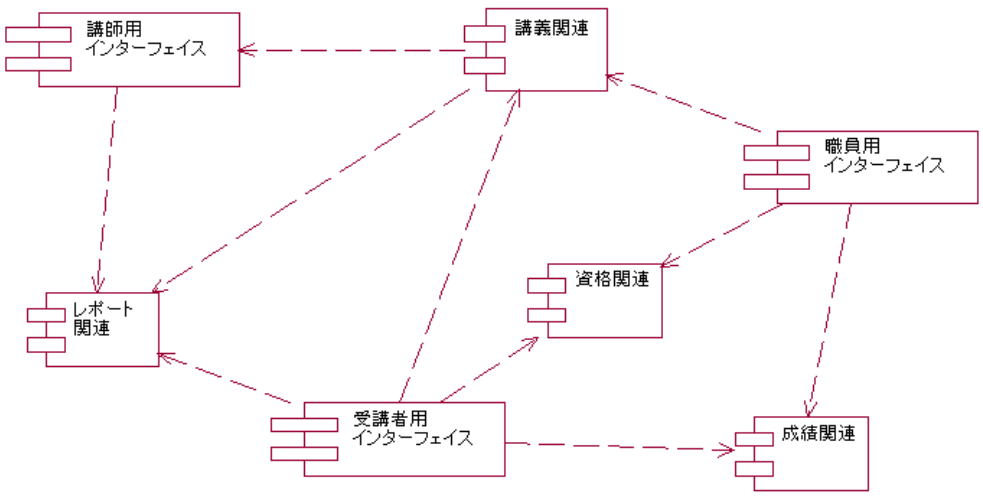


図 8: シーケンス図



各コンポーネントのクラス定義

・講師用インターフェイス ： 講師
・職員用インターフェイス ： 職員
・受講者用インターフェイス ： 受講者
・レポート関連 ： レポート課題, レポート
・成績関連 ： 個人別成績表, 科目別成績表, 成績明細
・講義関連 ： 開催科目, 募集案内, 会場, 受講申込, 受講申込書, 受講証, 受講者名簿
・資格関連 ： 資格, 資格認定証, 資格認定申請書

図 9: コンポーネント図

3 CBR と PBR の比較評価実験

3.1 概要

先に挙げた 2 つの手法によるレビュー実験を, 情報科学科 3 年生の学生 59 人に対して行った. この学生達は,「プログラム設計」の講義で UML について, また「ソフトウェア構成論」の講義でレビューについて学んでおり, それぞれに対する予備知識は持っていた. また, レビュー実験を行う約 1ヶ月前には,「ソフトウェア構成論」の講義において, 要求仕様書, ユースケース図, クラス図をもとにシーケンス図, コンポーネント図を作成するという演習も行っている.

レビュー実験は, 後述する 2 つのシステムをレビュー対象とし, 各手法, 役割に応じた UML 図, チェックリスト (シナリオ), バグ報告書を配布し, CBR, PBR の 2 つの方法でレビューを行った. その後, 配布した UML 図に検出したバグの該当箇所に印をつけたものと, 検出したバグに該当する UML 図名, そのバグを発見したチェックリスト (シナリオ) の項目, そしてそのバグの発見時刻を記録したバグ報告書を回収した.

3.2 実験詳細

実験に関する詳細を以下に記す.

3.2.1 レビュー対象システム

レビューの対象としたシステムには,「セミナ情報システム」と「医療情報システム」の 2 つを用いた. 各システムは要求仕様書, 5 種類の UML 図 (ユースケース図, クラス図, ステートチャート図, シーケンス図, コンポーネント図) により構成されている. また, PBR については各役割毎にレビューを行ったドキュメント数が異なっている. 各システムについて, それぞれのドキュメント数と役割ごとの割当てを表 2 に示す.

表 2: 各システムのドキュメント数及び割当て

システム, 役割 / ドキュメント	セミナ情報	医療情報	ユーザ	設計者	プログラマ
要求仕様書	1	1	○	○	○
ユースケース図	1	1	○	○	○
クラス図	1	1	×	○	○
ステートチャート図	5	1	○	×	×
シーケンス図	12	7	○	○	○
コンポーネント図	1	1	×	×	○

3.2.2 バグ

2 つのシステムそれぞれには, クラス図, ステートチャート図, シーケンス図, コンポーネント図に合計 15 個のバグを混入した. バグには 3 種類あり, 構文的なもの, 意味的なもの, 他の図との関連によるものがある. バグについての詳細を表 3 に示す. また, バグの例を図 10 図 から 12 に示す.

また, PBR については各役割について検出できるバグを限定しており, それについて以下に示す.

- ユーザ - バグ番号 4, 5, 6, 7, 8, 10, 11
- 設計者 - バグ番号 1, 2, 3, 8, 9, 12
- プログラマ - バグ番号 1, 2, 3, 9, 10, 12, 13, 14, 15

表 3: バグの種類 (両システム共通)

番号	UML 図名	分類 (構文的, 意味的, 関連)	内容
1	クラス図	構文的	クラス間の関連が無い
2	クラス図	意味的	冗長なクラスの存在
3	クラス図	構文的	多重度の定義漏れ
4	ステートチャート図	意味的	要求仕様に矛盾した状態の存在
5	ステートチャート図	構文的	状態の要素の欠如
6	ステートチャート図	意味的	状態の順序の違い
7	ステートチャート図	意味的	冗長な状態の存在
8	シーケンス図	意味的, 関連	必要なオブジェクトが存在していない
9	シーケンス図	意味的, 関連	矛盾したメソッド呼び出し
10	シーケンス図	構文的	メッセージ名の欠如
11	シーケンス図	意味的	必要なユースケースが実現されていない
12	シーケンス図	意味的	メソッド呼び出しの重複
13	コンポーネント図	関連	必要なクラスが存在していない
14	コンポーネント図	意味的	必要なコンポーネントが存在していない
15	コンポーネント図	意味的	冗長なコンポーネントの存在

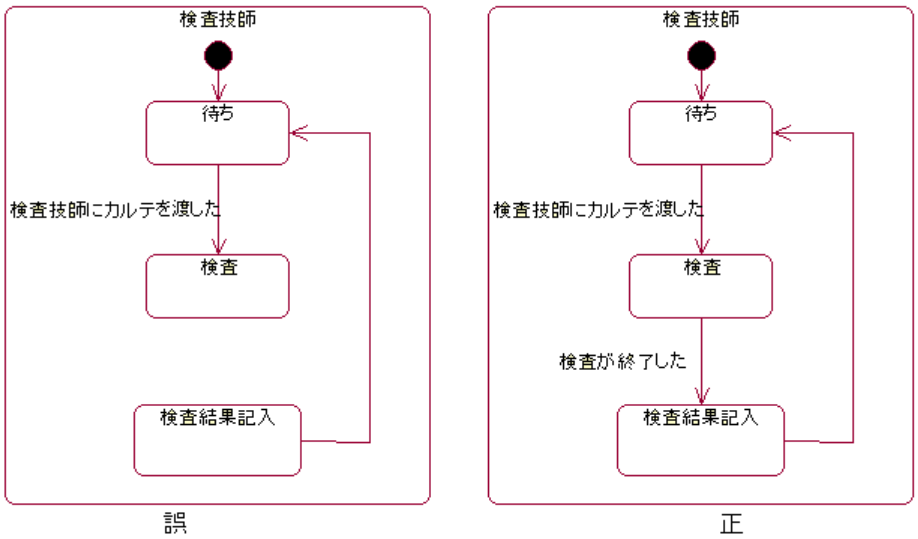


図 10: バグ番号 5(状態の要素の欠如)

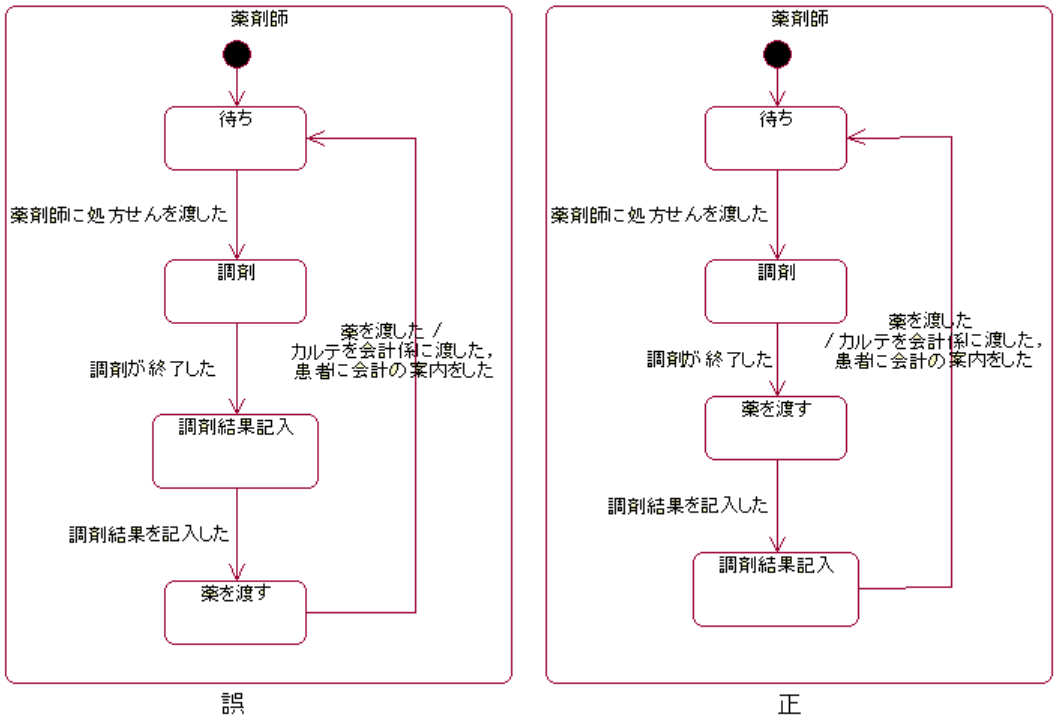


図 11: バグ番号 6(状態の順序の違い)

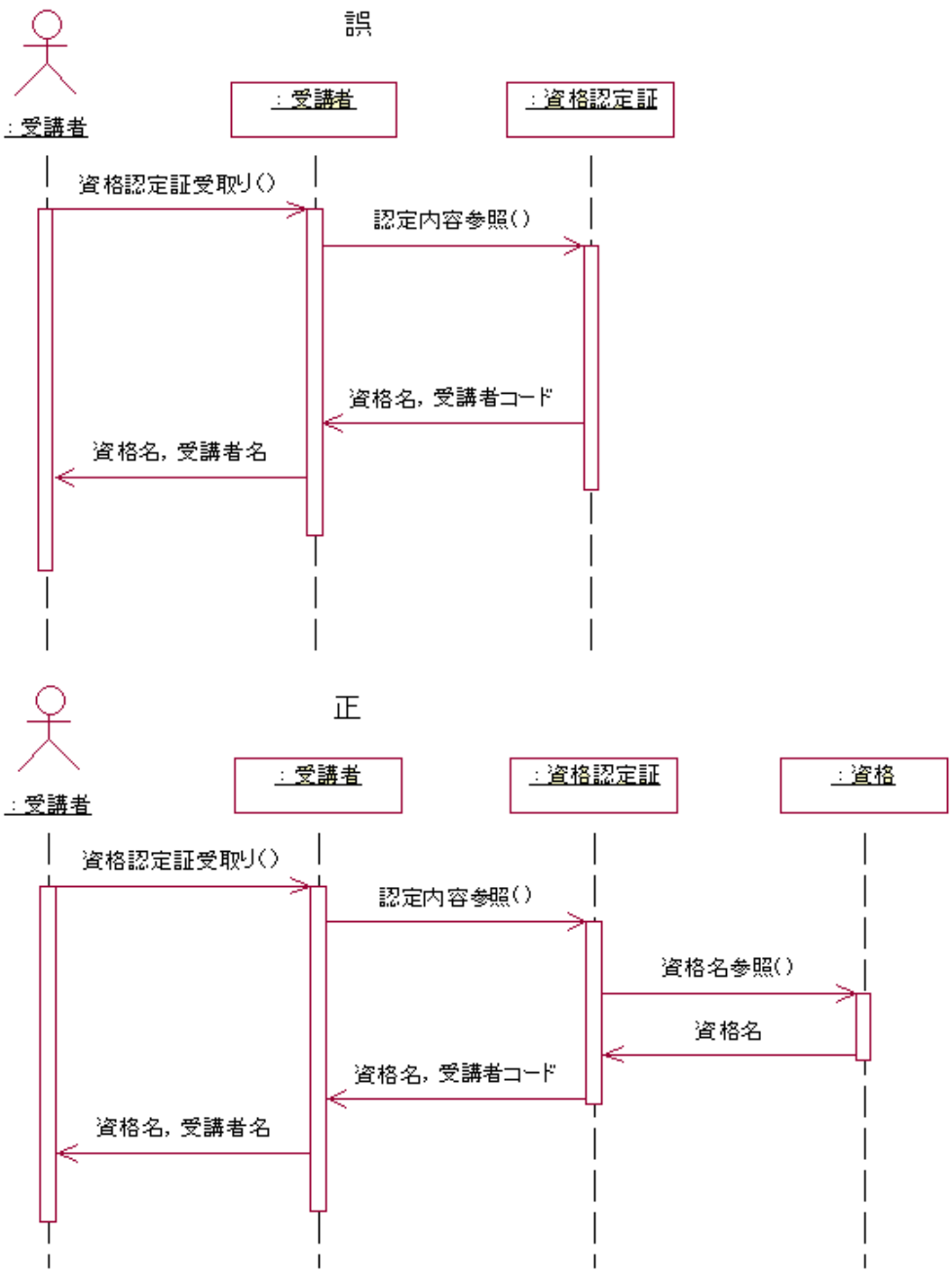


図 12: バグ番号 8(必要なオブジェクトが存在していない)

3.2.3 人数配分

実験に参加した 59 人の配分は表 4 の通りである.

表 4: 人数配分

	PBR			CBR
	ユーザ	設計者	プログラマ	
セミナ情報システム	7	6	6	11
医療情報システム	7	6	6	10

3.2.4 実験データ

この実験により収集したデータをまとめたものを図 13 に示す.

		人数	バグ			時間(分)	
			検出可能数	平均発見数	max/min	平均	max/min
セミナ情報システム	ユーザ	7	7	4.43	6/3	60.43	90/46
	設計者	6	6	5.00	6/4	65.50	80/51
	プログラマ	6	9	6.50	9/5	76.67	95/40
	CBR	11	15	10.55	13/8	74.64	90/62
医療情報システム	ユーザ	7	7	4.43	6/3	48.29	70/25
	設計者	6	6	3.83	5/3	59.17	73/30
	プログラマ	6	9	6.33	7/5	63.30	77/44
	CBR	10	15	10.50	12/8	70.10	94/60

図 13: 実験データ

これは, 提出された UML 図, バグ報告書に基づいて各値を算出したものである. 各システム, 各手法毎に, バグについては検出可能数, 実際の検出数の平均, 最大値と最小値を記し, 時間についてはレビュー時間の平均, 最大値と最小値を記した.

4　実験データの分析と考察

実験で得られたデータを以下の 2 つの方向から分析を行った.

4.1　個人データ分析

個人データについては, バグの検出率 [(検出バグ数)/(検出可能バグ数)] を以下の 3 つの観点から計算し, 分析を行った.

4.1.1　手法による違い

結果を図 14 に示す.PBR の各役割に付随する CBR での値は, 各役割で検出可能なバグに合わせて CBR で算出したものである. CBR 全体での平均バグ検出率が 70.16%, PBR 全体での平均バグ検出率が 69.10%であった.

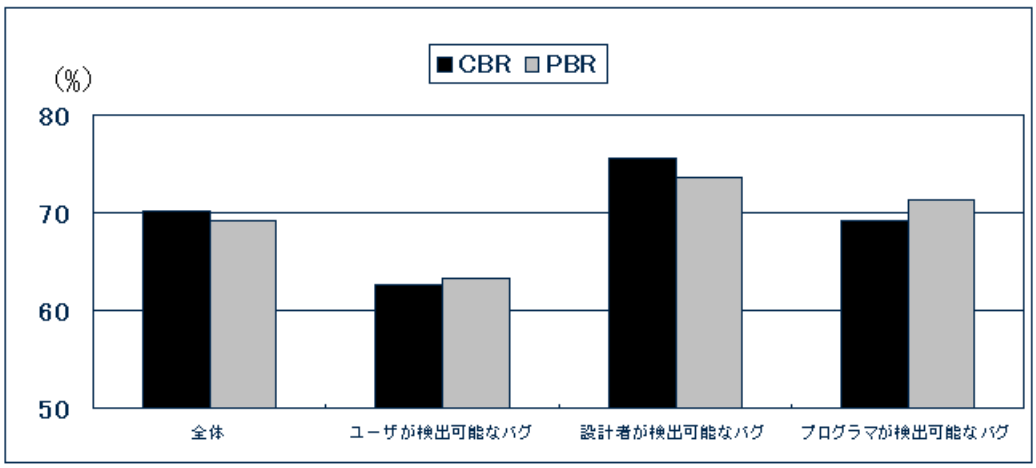


図 14: バグ検出率 (手法)

4.1.2　バグの種類別

結果を図 16 に示す. 構文的なバグについては CBR, 意味的なバグや他の図との関連に関するバグについては PBR がより高い検出率を示した.

4.1.3　UML 図別

結果を図 15 に示す. クラス図においては CBR に, 残りの 3 つの図では PBR に高い検出率が見られた.

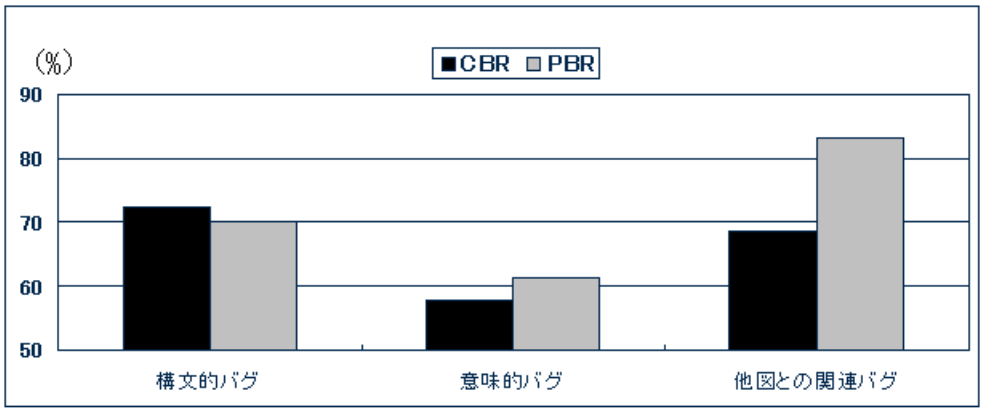


図 15: バグ検出率 (バグの種類)

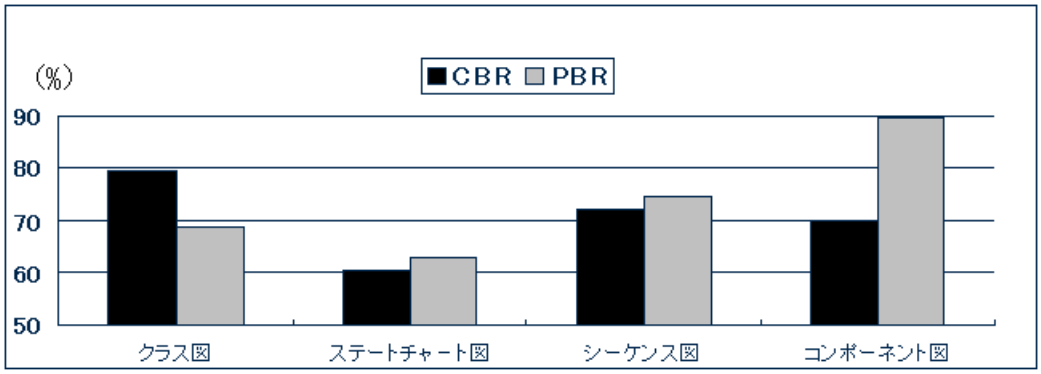


図 16: バグ検出率 (UML 図)

4.2 チームデータ分析

一般にレビューは, 対象となるプロダクトに対し複数の人間がチームとして行う. そこで, 被験者を組み合わせて仮想的にチームを作り, その上でのデータ分析を行った.

4.2.1 チームの組み方

チームを作るにあたっては, 2 つの基準を設け, その基準に従って組み合わせた.

1. 検出可能バグ数 (基準 A とする)
   今回行った実験では, 検出可能なバグ数がそれぞれ異なっていた. CBR は一人あたり設定したバグ 15 個すべてを検出可能であるのに対し, PBR ではユーザ, 設計者, プログラマの 3 人を合わせることで初めて 15 個検出できるように設定していた. このことから, 各システムにおいて以下の方式でチームを組ませた.

   - CBR - 1 チーム 1 人
   - PBR - 1 チーム 3 人 (ユーザ, 設計者, プログラマから 1 人ずつ無作為に選出)

2. 人数 (基準 B とする)
   これは単に 1 チームあたりの人数をそろえて組ませたものである.

   - CBR - 1 チーム 3 人 (無作為に選出)
   - PBR - 1 チーム 3 人 (ユーザ, 設計者, プログラマから 1 人ずつ無作為に選出)

4.2.2 グループ化

前述のようにチームを組ませた後, さらにグループ化を行った.
まず基準 A の組み方では, CBR はセミナ情報システムでは 11 チームで 1 グループとし, 医療情報システムでは 10 チームで 1 グループとした. PBR については 1 つの組み合わせ方でチームを組ませた場合に 6 チーム出来るので, それを 1 グループとした.
次に基準 B の組み方では, CBR は 1 つの組み合わせ方でチームを組ませた場合に 3 チーム出来るので, それを 1 グループとした. PBR については基準 A の場合と同様である. 基準 A でのグループ化を図式化したものを図 17 に示す.

4.2.3 データ算出

前述のようにグループ化を行った後, バグ検出数, レビュー時間についてデータ算出を行った. 詳しい算出方法を基準別で以下に示す.

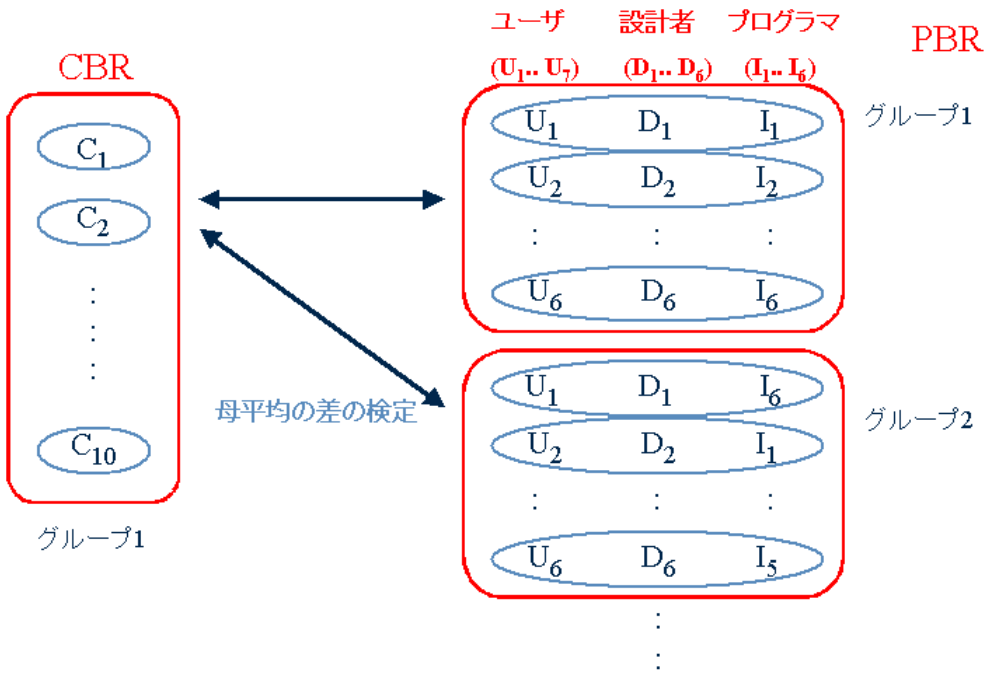


図 17: 基準 A でのグループ化 (楕円部分がチーム, 四角の部分がグループを表す)

1. 基準 A

   CBR では, 1 チーム 1 人なので, メンバーのそれぞれの検出数の平均値をグループのバグ検出数, レビュー時間の平均グループの時間とした.

   PBR では, 15 個のバグについて 3 人の論理和をとり, チームのバグ検出数とした. レビュー時間については 3 人の最大時間をチームの時間とした. その後 6 チーム分の平均値をグループでのバグ検出数, レビュー時間とした.

2. 基準 B

   CBR では, 15 個のバグについて 3 人の論理和をとり, チームのバグ検出数とした. レビュー時間については 3 人の最大時間をチームの時間とした. その後 3 チーム分の平均値をグループでのバグ検出数, レビュー時間とした.

   PBR については基準 A と同様である.

4.2.4 データ比較

このようにして各グループでのデータを算出し, グループ間の比較を行った. 比較は単に検出数と時間を比べ, 有効な方を勝利としてその勝ち数を算出した方法と, バグ検出数について有意水準 2.5%で母平均の検定を行った. 前者の比較方法の結果を基準 A は表 5 に, 基

準 B は表 6, 表 7 に示す.

表 5: 比較結果 (基準 A)

	バグ検出数			レビュー時間			総比較回数
	CBR	PBR	等しい	CBR	PBR	等しい	
セミナ情報	893	3,615,154	12,753	1,524,642	2,050,817	53,341	3,628,800
医療情報	352	3,611,280	17,168	1,209,600	2,384,640	34,560	3,628,800

表 6: 比較結果 (基準 B その 1)

	バグ検出数		
	CBR	PBR	等しい
セミナ情報	54,753,326,688	585,743,712	544,449,600
医療情報	10,160,482,176	8,064	149,760

表 7: 比較結果 (基準 B その 2)

	レビュー時間			総比較回数
	CBR	PBR	等しい	
セミナ情報	27,809,497,152	27,965,566,656	108,456,192	55,883,520,000
医療情報	34,972	10,154,262,899	6,342,129	10,160,640,000

また検定を行った結果, 基準 A では PBR に, 基準 B では CBR にそれぞれ有効性が確認された.

4.3 考察

これまでの結果から, 個人データ分析, チーム分析両方について考察を記す.

- 個人データ
  - 手法による違い
    分析結果から, 今回の実験では, 個人データにおいて CBR と PBR に特に明確な差は見られなかった.

- バグの種類別
  分析結果から, CBR はプロダクトすべてを単に yes/no で答える形式であることから, プロダクトを表層的にチェックしているため, 構文的なバグは見つけやすいと考えられる. また, PBR はチェックするプロダクトが限られていることと, シナリオの中で他の図との一貫性をチェックする作業が含まれていることから, プロダクトに対しより深くチェックを行うことができ, 意味的なバグ, 他の図との関連によるバグがより多く発見できたと考えられる.
- UML 図別
  分析結果から, その原因として, クラス図に含まれていたバグに構文的なものが多かったことと, コンポーネント図に含まれていたバグに意味的なバグ, 他の図との関連によるバグが多かったためであると考えられる.

- チームデータ
  分析結果から, チームでの比較においては, Laitenberger らの実験のように, どちらの手法が有効であるかを結論付けることは出来なかった. これには以下の点が原因ではないかと考えられる.
  - 実験方式の違い
    Laitenberger らの実験では 2 つのシステムを用いて, レビュー実験を 2 日に渡って行い, 参加者は 1 日目と 2 日目でシステム, 手法をいれかえて行っていた. つまり, 1 日目にシステム A を CBR でレビューした者は, 2 日目にはシステム B を PBR でレビューを行っていた. 本研究における実験では 1 日でしか行っていなかった.
  - 埋め込んだバグについて
    今回埋め込んだ合計 15 個のバグについてだが, 事前に学んではいても実際のレビューには慣れていない学生達であったにも関わらず, 両手法において平均的に全体の検出率が高かったため, 手法による違いを見るために有効なバグではなかったのではないかと考えられる.

5　まとめと今後の課題

本研究では, UML で記述された設計仕様書を対象とし, 2 つのレビュー手法 CBR と PBR を用いてレビュー実験, 得られた実験データの分析を行い, 両手法の比較評価を行った. 実験では情報科学科の 3 年生 59 人を被験者とし, レビュー対象として 2 つのシステムの UML 図を用いた. 被験者はあらかじめ複数のバグを埋め込んだ UML 図を，指定されたレビュー手法でチェックし, 検出したバグ, 検出時刻等を報告した. 分析では, 個人データでのバグ検出率の分析と, チームに分けてバグ検出数とレビュー時間の比較を行った.

その結果, 構文的なバグ検出には CBR, 意味的なバグ検出や UML の図間の関連や一貫性をチェックするという点においては PBR に, それぞれ有効性が確認された.

今後の課題としては,

- チーム間の比較におけるより詳細な分析
- 更なる追証実験によるデータ収集

などが挙げられる. 前者については, 被験者にアンケートをとりその結果を分析に反映させることや, チームの組み合わせに新たな基準を設ける等, 現在も思案, 分析中である. 後者については, 今後も, 学生を対象とし, 考察を踏まえた環境を設定した新たなレビュー実験等を予定している.

謝辞

本研究において, 常に適切な御指導および御助言を頂きました大阪大学 基礎工学研究科 情報数理系専攻 ソフトウェア科学分野 井上 克郎 教授に深く感謝致します.

本研究において, 常に適切な御指導および御助言を頂きました 同 楠本 真二 助教授に深く感謝致します.

本研究において, 常に適切な御指導および御助言を頂きました 同 松下 誠 助手に深く感謝致します.

本研究において, 常に適切な御指導および御助言を頂きました 同博士後期過程 1 年 Giedre Sabaliauskaite さんに深く感謝致します.

最後に, その他様々な御指導, 御助言を頂いた大阪大学 基礎工学研究科 情報数理系専攻 ソフトウェア科学分野 井上研究室の皆様に深く感謝いたします.

参考文献

[1] V.R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, M.V. Zelkowitz, “The Empirical Investigation of Perspective-Based Reading”, Empirical Software Engineering, I:133-164, 1996.

[2] B.W. カーニハン, D.M. リッチー, 石田 晴久 [訳], “プログラミング言語 C”, 共立出版, 1989.

[3] G. Booch, J. Rumbaugh, I. Jacobson, “The Unified Modeling Language User Guide”, Addison Wesley Longman, Inc., 1999.

[4] T. Gilb, D.Graham, 伊土 誠一, 富野 壽 [監訳], “ソフトウェアインスペクション”, 共立出版, 1999.

[5] IEEE, “IEEE Standard for Software Reviews and Audits”, ANSI/IEEE Std:1028-1988, 1988.

[6] 伊藤 潔, 廣田 豊彦, 冨士 隆, 熊谷 敏, 川端 亮, “ソフトウェア工学演習”, オーム社, 2001.

[7] N. Juristo, A.M. Moreno, “Basics of Software Engineering Experimentation”, Kluwer Academic Publishers, 2001.

[8] 鍵和田 京子, 石村 貞夫, “よくわかる卒論・修論のための統計処理の選び方”, 東京図書, 2001.

[9] O. Laitenberger, C. Atkinson, M. Schlich, K. El Emam. “An experimental comparison of reading techniques for defect detection in UML design documents”, The Journal of Systems and Software, 53:183-204, 2000.

[10] O. Laitenberger, J.M. DeBaud, “An encompassing life cycle centric survey of software inspection”, The Journal of Systems and Software, 50:5-31, 2000.

[11] O. Laitenberger, C. Atkinson, “Generalizing perspective-based inspection to handle object-oriented development artifacts”, Proceedings of the 21st International Conference on Software Engineering, 2000.

[12] O. Laitenberger, K. El Emam, T.G. Harbich, “An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective-Based Reading of Code Documents”, IEEE Transactions on Software Engineering, V.27, No.5:387-421, May 2001.

[13] A. Porter, L.G. Votta, V. Basili, “Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment”, IEEE Transactions and Software Engineering, V.21, No.6, 1995.

[14] B. Regnell, P. Runeson, T. Thelin, “Are the Perspectives Really Different? Further Experimentation on Scenario-Based Reading of Requirements”, Technical Report LUTEDEX (TETS-7172):1-40, 1999. Department of Communication Systems, Lund University.

[15] J.Schmuller, 多摩ソフトウェア [訳], 長瀬 嘉秀 [監訳], “独習 UML”, 翔泳社, 2000.

[16] T. Thelin, P. Runeson, B. Regnell, “Usage-based reading - an experiment to guide reviewers with use cases”, Information and Software Technology, 43:925-938, 2001.

[17] 土田 昭司, “社会調査のためのデータ分析入門”, 有斐閣, 1994.

[18] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, A. Wesslen, “Experimentation in software engineering: an introduction”, Kluwer Academic Publisher, 2000.

付録

付録として, 以下のものを添付する.

- 要求仕様書, 及び UML 図
  - セミナ情報システム
  - 医療情報システム
- チェックリスト (シナリオ), 及びバグ報告書
  - CBR 用
  - PBR(ユーザ) 用
  - CBR(設計者) 用
  - CBR(プログラマ) 用