3B-5

UMLを対象としたソフトウェアオーバーホール手法とその環境

内田眞司† 山本舜† 里中健太郎‡

奈良工業高等専門学校 情報工学科† 奈良工業高等専門学校専攻科 電子情報工学専攻‡

1. はじめに

ソフトウェアレビューとはソフトウェア開発の各工程において作成される成果物に対して，作成者と他の開発者間で成果物の中身を確認し，最終的に成果物にバグが残らないようにする作業を指す[1]．一般に，バグの発見が遅れるほど，その修正にかかるコストは大きくなるため，開発工程の初期段階でバグを発見することでソフトウェアの品質や生産性を向上させることが可能となる．

ソフトウェア開発工程において生成される成果物には，自然言語やプログラム言語など言語で記述されているものの他に，様々な種類のモデリング手法を用いてソフトウェアの体系を視覚的に記述しているものがある．例えば，構造化分析設計手法におけるDFDやモジュール階層図，オブジェクト指向分析設計法で用いられるUML等が該当する．ソフトウェアレビューにおいてはこれらのモデルそのものや，モデルを含んだ成果物も対象となる．しかし，これまでに提案されているレビュー手法はソースコードを含むテキストベースの成果物を対象としており，モデルやそれを含む成果物に対する体系的なレビュー手法は提案されていない．

従来のレビュー手法であるCBR(Checklist-Based-Reading) と PBR(Perspective-Based-Reading)を，UMLを含む設計仕様書に対して用いたレビュー実験では，構文的なバグの検出にはCBR，意味的なバグの検出にはPBRに有効性が確認されたと報告されている[2]．しかし，意味的なバグの検出率は，60%未満であった．意味的バグは，成果物をより深くチェックすることで発見できたと指摘されており，検出率を向上させるためには成果物を深くチェックする体系的な仕組みが必要となる．

本稿では，UMLで記述された設計仕様書を対象としたレビューを目的とするソフトウェアオーバーホール手法とその作業を支援する環境について述べる．ソフトウェアオーバーホール手法とは作業者がソフトウェアを理解するプロセスを計測する手法[3]で，ソフトウェアの分解と再統合から構成される．UML図を分解・再統合する過程が，ハードウェア分野で使われるオーバーホール手法から類推されるように，深層的なレビュー手法となり，仕様とUML図との間の矛盾(バグ)を浮き彫りにする．レビュー作業にソフトウェアオーバーホール手法を適用することで，従来のレビュー手法では実現不可能な体系的なレビューを実現する．

Software Overhaul for the diagram of UML and its environment

†ShinjiUCHIDA, †Shun YAMAMOTO and ‡Kentaro SATONAKA

†Information Engineering, Nara National College of Technology

‡Advanced Electronic and Information Engineering, Faculty of Advanced Engineering, Nara National College of Technology

2. UMLを対象としたソフトウェアオーバーホール手法

2.1. 概要

ソフトウェアオーバーホール手法の概念を図 1 に示す．図中，丸印はソフトウェアのコンポーネント(関数やステートメント)を示しており，矢印はコンポーネント間の依存関係を示す．ここでUMLのアクテビティ図を対象とした場合，コンポーネントは図を構成するアクションノードやデシジョンノードなどの要素となり，矢印は要素間の遷移を示す矢印となる．ソフトウェアの分解では，ソフトウェアをコンポーネントに分解する．具体的にはコンポーネント間の依存関係を破棄し，ランダムに配置する．再統合では，分解されたコンポーネントの依存関係を，仕様書などを足がかりにして，作業者が元通りに再統合する．オーバーホールの対象となるUML図にバグがある場合，再統合されたUML図との間に違いが表層化される．

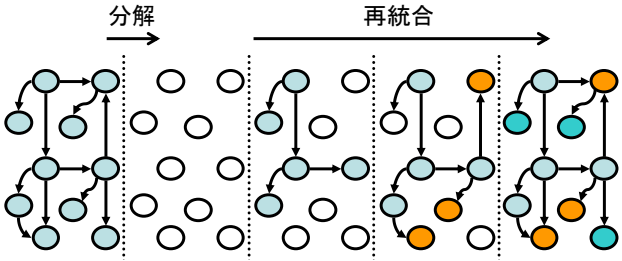

図 1 ソフトウェアオーバーホールの概念

2.2. 環境

ソフトウェアオーバーホールでは，PC上に用意されたオーバーホールツール，オーバーホールを行うUML図の仕様を表したドキュメントが作業者に用意される．ここで言うドキュメントとは，自然言語で書かれた対象となるUML図のシナリオである．作業者は，ツールによってコンポーネント単位で分解されたUML図の再統合を行う．ドキュメントを読みコンポーネントを理解しながら，バラバラに分解されたUML図を元通りに戻す．その際ツールでは再統合の過程を履歴として収集している．再統合が完了した時点で，コンポーネントの位置がもとどおりに構築されたかをツールにより判定する．すべてのコンポーネントを正しく統合することができたなら，オーバーホールは終了する．図 2 に試作したオーバーホールツールの外観を示す．ツールはUMLエディタであるVioletUMLEditor[4]に分解機能と判定機能を追加する方法で試作した．図は，対象UMLをアクテビティ図として，ツールが分解した直後を示している．

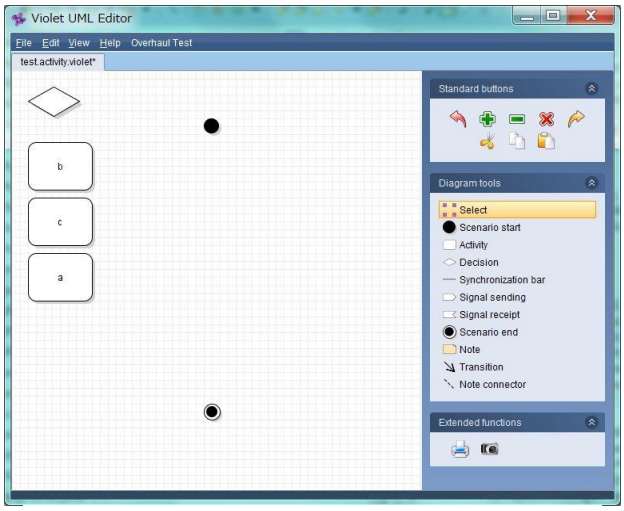

図 2　オーバーホールツール外観

3. 適用例

2.2 で述べた環境の適用例を述べる．作業者は奈良高専専攻科学生 1 名で約 6 年のプログラミング経験があった．対象システムは「休業日数計算システム」で，その処理の流れ記述したアクテビティ図を対象とする．この図には，表 1 に示す 3 個の意味的なバグが混入されていた．作業者にはオーバーホール作業を通して，発見したバグを指摘(レビュー作業)してもらった．

適用した結果，作業者は全てのバグを指摘することができた．レビュー作業に要した時間は約 32 分で，オーバーホール作業には約 24 分を要した．

作業後に作業者へインタビューしたところ，番号 1 と番号 3 のバグについては，再統合後のツールによる判定を通して発見したと答えた．対象のUML図と作業者が再構築したUML図で相違点がツールの判定機能により表層化されたため，仕様書との矛盾に気がついたと考えられる．またオーバーホール作業の履歴を分析した結果，バグの存在するコンポーネントについて作業者は誤って再統合していることが分かった．一方，番号 2 のバグについては再統合を繰り返すうちに，仕様を深く理解することでバグの発見に至ったと答えた．以上から，本研究の手法がUML図を含む仕様書を対象としたレビューに有用であることが示唆された．

表 1 バグの種類と発見時間

No	バグの内容	発見時間
1	仕様と異なる分岐条件	約24分
2	必要な処理の欠如	約32分
3	処理手順の違い	約20分

4. まとめと今後の課題

本稿では，UMLで記述された設計仕様書を対象としたレビューを目的とするソフトウェアオーバーホール手法とその作業を支援する環境について述べた．試作したツールを用いた適用実験では，UML図中に埋め込まれた意味的なバグを全て発見することができた．今後の課題としては，試作ツールの改良，従来手法との比較実験などが挙げられる．

参考文献

[1]S.Kusumoto, A.Chimura, K.Matsumoto, T.Kikuno, Y.Mohri : A Promising Approach toTwo-person Software Review in EducationalEnvironment, Journal of Systems & Software,Vol. 40, No. 2: 115-123, February, 1998.

[2] 松川文一，Giedre Sabaliauskaite，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，pp.67-74, August 28-30, 2002.

[3]内田眞司，島 和之，武村泰宏，松本健一 : ソフトウェアオーバーホール手法の実験的評価，情報処理学会論文誌，Vol.49，No.3，pp.1330-1340，March 2008.

[4] Violet UML Editor : http://alexdp.free.fr/violetumleditor/page.php