Vol.2010-SE-167 No.30
2010/3/19

バグ報告の単語出現頻度に着目したチェックリスト作成の試行

渡邊正隆[†1]　森崎修司[†1]　松本健一[†1]

ソフトウェアの品質向上を目的として，欠陥を発見につながる具体的方法を記述したチェックリストの効率的な作成を目指し，過去に報告されたバグを抽象化する手順を提案し，オープンソースソフトウェアに適用した結果を報告する．チェックリストは熟練者が過去に起こっているバグを元に抽象化して作成される方法が一般的であるが，本稿では，チェックリストの作成方法として過去のバグ報告に含まれる単語に着目し，単語の出現頻度によりバグ報告の関連性を明らかにし，バグを抽象化する．抽象化したバグをもとにチェックリストができるかどうかを調査するために，Eclipse のプラグインである GEF(Graphical Editing Framework) を対象に過去のバグ報告に含まれる単語の出現頻度からバグの抽象化を試み，それよりも新しいバージョンのバグが未然に防げたかどうか調査したところ，いくつかの単語でバグを未然に防げていた可能性があったことを確認できた．

Experimental evaluation of creating checklist by word occurrence in bug report

MASTAKA WATANABE,[†1] SHUJI MORISAKI[†1]
and KEN-ICHI MATSUMOTO[†1]

This paper proposes an abstraction procedure of bug description in order to create software quality checklist. We applied the procedure to a bug repository of an open source project. Checklist is generally created by experts by using their knowledge. Word occurence in bug repository is used for abstraction for bugs. In order to evaluate whether bug abstraction can be checklist, we obtain abstract bugs from an open source Eclipse plug-in GEF. Abstracted bugs from GEF version 2.x are examined to prevent bugs in version 3.x. The result shows some bugs in version 3.x could have prevent by using abstracted bugs from version 2.x.

1. はじめに

ソフトウェア品質を向上することを目的とした検証活動としてソフトウェアレビュー，ソフトウェアテストが実施されている．ソフトウェアに混入された多くの不具合は，開発中に実施されるこれらの検証活動により除去される．特にソフトウェアレビューは，プログラムが実行できる段階にない場合でも実施できるため，不具合の早期発見が可能である．

しかしながら，プログラムの動作が期待どおりかどうかを 1 つずつ確認しながら進めるソフトウェアテストと異なり，ソフトウェアレビューでは進め方に決まった方法がない．そのため，リーディングテクニックと呼ぶ読み進め方が多数提案されている．[4)5)6)] また，それら技法の間での評価が数多く報告されている．[7)8)]

ソフトウェアレビューで使われるチェックリストはリーディング技法の中でも基本的かつ大きな効果が得られるものとして認められている．どのようなレビューアが実施しても同程度の結果が得られること，チェックリスト自体の再利用が容易であることがその理由であると考えられる．また，既存のチェックリストの改善方法も提案されている[3)]．しかしながらチェックリストを作成する方法が確立されているとは言い難く，熟練者の経験から作成されているのが現状である．

本稿では，同一，あるいは類似のソフトウェアを開発する際に自然言語で記述され，蓄積された過去のバグ報告に注目し，それら報告からバグを抽象化しチェックリストを作成することを目的し，その手順を提案し，その手順をオープンソースソフトウェアに適用し，その有効性を確認する．具体的には，過去のバグ報告の症状，再現方法，修正方法の記述から各々のバグの要約を作成する．次に複数の要約に含まれる単語を抽出しその出現頻度を計数する．出現頻度の高い単語を含む複数のバグ報告を抽象化し，抽象化バグを得，チェックリストとしてバグを未然に防げるかを確認する．

以下，2 章では抽象化手順を述べる．3 章では 2 章の手順に従い，オープンソースを対象として実施した試行の概要を述べ，4 章でその結果を述べる．5 章はまとめと今後の課題である．

†1 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

Vol.2010-SE-167 No.30
2010/3/19

2. バグ抽象化の手順

ソフトウェアレビューやコードレビューの際に用いるチェックリスト項目作成の為にバグ報告の抽象化を行う．具体的手順を図 1 に示す．

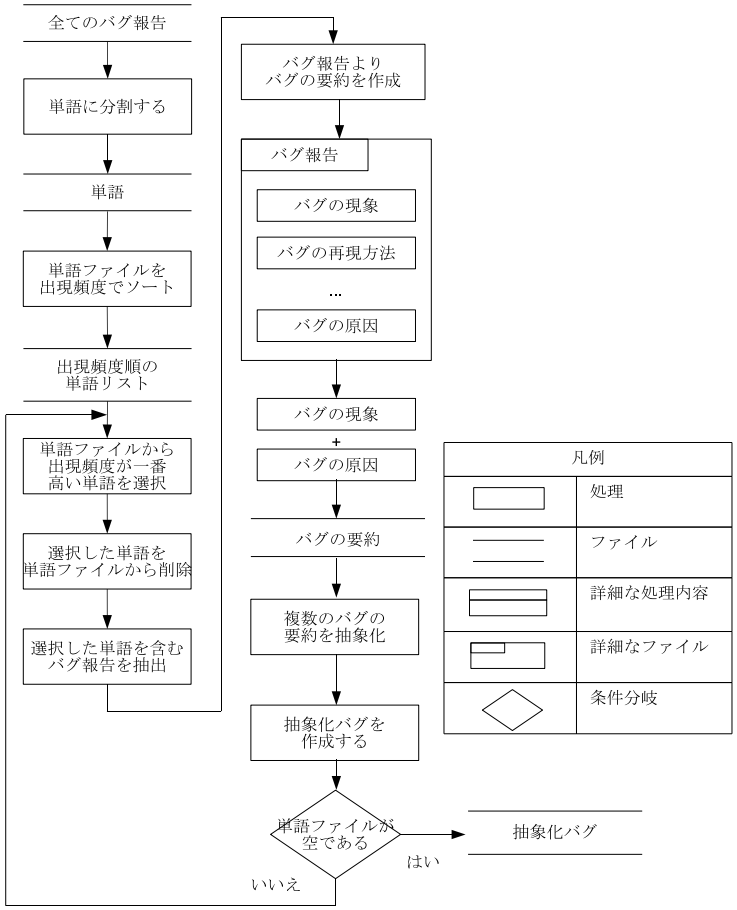

図 1 バグ抽象化の流れ

以降では具体例を示しながら図 1 を説明する．

(1) すべてのバグ報告から単語抽出

すべてのバグ報告の中から単語の出現頻度を計数するために，それぞれのバグ報告から単語抽出する．

(2) 単語の出現頻度の計数

(1) より分割された単語の出現頻度を計数して高い出現頻度別に並び替える．並び替えた単語より出現頻度別単語リストを作成する．

(3) バグ報告の抽出 (3) で作成した単語リストの上位にある単語を含むバグ報告をすべてのバグ報告の中から抽出する．

(4) バグ要約の作成

ひとつのバグ報告は，バグの現象と原因，それ以外の文章から構成されている．バグの現象はバグ報告の最初のほうに記録されておりバグの原因は最後のほうに記録されていることが多い．そこでバグ報告の最初と最後を抽出し接続詞を入れてバグ要約を作成する．バグ要約を作成する手順の一例を以下に示す．

- 現象として“ プログラムの動作が終了しない ”がある．
- 原因として“ カンマとドットを間違えてループを抜けられない ”がある．
- 作業者によって差がでないように，現象と原因を接続詞等で繋いでバグ要約を作成する．
- 上記の例では“ カンマとドットを間違えてループを抜けられないため，プログラムの動作が終了しない．”となる．

(5) バグの抽象化

(4) で作成したバグ要約より共通点が多いと思われるバグ要約を抽象化して抽象化バグを作成する．抽象化バグを作成する手順の一例を以下に示す．

- 現象として“ プログラムの動作が終了しない ”がある．
- 原因として“ カンマとドットを間違えてループを抜けられない ”がある．
- 抽象化する者によって差がでないように，現象と原因を接続詞等で繋いでバグ要約を作成する．
- 上記の例では“ カンマとドットを間違えてループを抜けられないため，プログラムの動作が終了しない．”となる．
- このバグ要約を抽象化すると“ ループ条件を確認していないバグ ”となる．

(6) 抽象化バグのグルーピング

作成された抽象化バグを似たようなもの同士でグルーピングを行う．グルーピングする際に，抽象度が高すぎず多くのバグ報告をまとめられた場合，良い抽象化と言える．

3. 試行概要

上記で提案した手法を用いて，総合開発環境の Eclipse のプラグインである GEF(Graphical Editing Framework) のバグ管理システムに登録されているバグ約 400 件について試行を行った．具体的な手順については以下の通りである．

3.1 試行手順 (抽象化)

(1) すべてのバグ報告文章から単語に分割
今回試行した対象バグ管理システムのバグ報告文章は英語で記述されているため，すべてのバグ報告文章を単語に分割した．バグ報告文章中にソースコードが含まれる場合があるが，この段階では考慮しないものとした．また，バグ報告文章を単語に分割するためスペースごとに区切るという方法を用いた．

(2) 単語の選別
(1) にて得られた単語のそれぞれについて単語ごとに出現頻度を計数した．表 1 は取り出した単語一覧の上位 20 単語である．
表 1 の通り単語に分割しただけでは，前置詞や接続詞などの上位になるこれらの単語を除くため単語の選別を行った．また，GEF は GUI アプリケーションに関係するので GUI アプリケーションに関連すると思われる単語を残した．選別を行った結果を表 2 に示す．

(3) バグ報告の抽出
選別した単語を用いてバグ報告の抽出を行う．各単語に対して抽出できたバグ報告件数は表 3 のとおりである．

(4) バグ要約の作成
(3) で抽出したバグ報告について今回の試行では表 3 の下位にある単語である“ size ”,“ connection ”,“ select ”,“ view ”の 4 つを選びバグ報告よりバグ要約の作成を行った．

(5) バグの抽象化とグルーピング
2 章で述べた手順で抽象化を行う．

4. 試行結果

バグ報告の単語出現頻度に着目したチェックリスト作成の試行を行った結果，400 件のバ

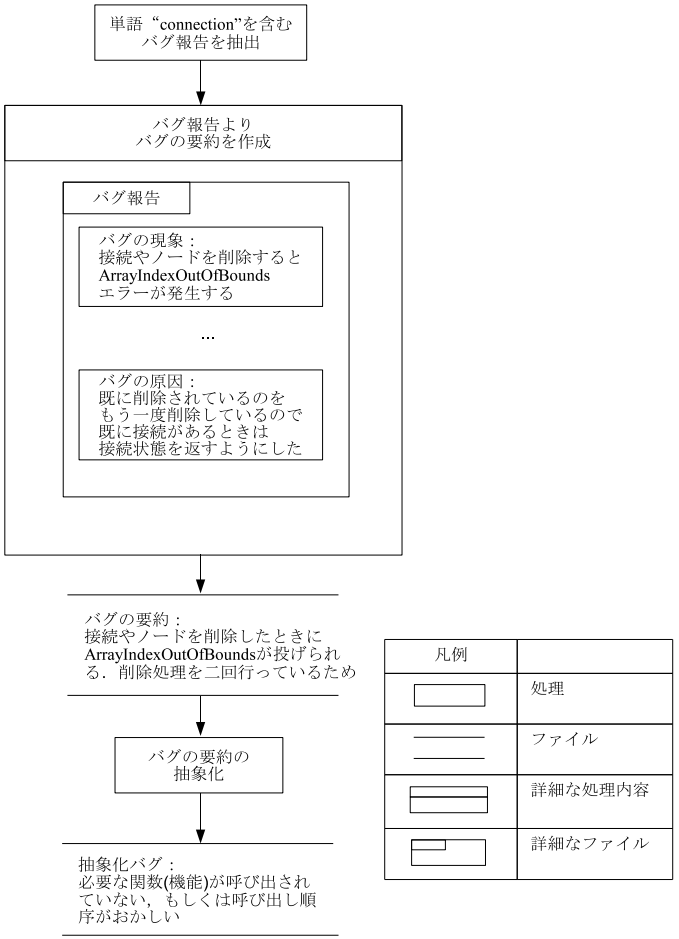

図 2 バグ抽象化の具体的な流れ

表 1 出現頻度上位 20 単語

単語	出現頻度 (回)
the	3121
at	1481
to	1316
is	961
in	954
a	917
and	755
of	564
this	519
it	493
not	459
be	452
=	430
i	428
for	410
that	392
if	369
new	354
>	339
on	328

表 2 出現頻度の上位 20 単語 (前置詞等を除く)

単語	出現頻度 (回)
palette	242
line	140
editor	133
logic	115
figure	110
import	109
text	109
return	98
create	96
diagram	91
file	59
label	73
edit	89
size	80
mouse	86
connection	72
point	65
select	63
view	56
zoom	68

表 3 単語別バグ抽出件数

単語	抽出件数
palette	104
line	48
editor	44
logic	75
figure	84
import	12
text	56
return	50
create	68
diagram	54
file	32
label	31
edit	44
size	43
mouse	45
connection	28
point	36
select	44
view	31
zoom	29

グ報告から 3.1 の手順により 4 つの単語について試行を行った．単語”connection”については表 4 のようになる．紙面の関係上, 単語“ connection ”についてのバグ要約のみを記載している．表 5,6,7,8 はバージョン別の抽象化バグの該当件数を示すものである．

4.1 バグの抽象化

表 4 より ver2.x のバグ報告であるバグ報告 ID37145,37201,37611,39437 についてバグの抽象化を試みる．バグ報告 ID37145 では“ 接続やノードを削除したときに ArrayIndexOutOfBounds が投げられる．削除処理を二回行っているため．”となっている．このバグ報告を抽象化する場合，前半部分のバグの現象ではなく後半部分の原因に注目する．“ 削除処理を二回行っているため ”という文章を抽象化すると“ 不要な処理が呼び出されている．”となる．

次にバグ報告 ID37201“ NullConnectionRouter が接続部分の位置を更新しない ”というバグ報告の抽象化を行う．このバグ報告にはバグの現象のみしか記載されていないためバグの現象の抽象化を行うと“ 更新する機能 (関数) を呼び出していない ”となる．同様にバグ

表 4 単語“ connection ”バグ報告のバグ要約

バグ報告 ID	バージョン	バグ要約
30795	2.1	EditParts では D&D ができるが，ConnectionEditParts ではできない．関数のオーバーライドに問題がある
33366	2.1	v2.1 で Ctrl+\ のキーイベント処理が変わったように思える．特殊なキーコードなので GraphicalViewerKeyHandler.acceptConnection へ’u001c’ を加えてください
35312	2.1	マーキーツールと接続作成ツールがおかしい．デフォルトで unloadWhenFinished されるはずがない
36221	2.1	ソースオブジェクトと目的オブジェクトを繋いでソースオブジェクトを削除すると繋いだ部分のダミーが残る.EditPartListener に setconnectionSource 関数を追加し，それに関するイベントリスナーを削除した
36466	2.1.1	接点の位置調節を行った後，カンマキーに割れ当てられた機能が働かない
37145	2.1.1	接続やノードを削除したときに ArrayIndexOutOfBounds が投げられる．削除処理を二回行ってるため
37201	2.1.1	NullConnectionRouter が接続部分の位置を更新しない
37446	2.1	graphical viewer で接続部分を選択した後 treeviewer を選択するとマウスイベント (mousemove) がすべて treeviewer に投げられるようになる
37611	2.1	ソースノード削除時のフィードバック表示が削除されずに残る
37613	2.1	接続部分が主要 (オブジェクト) を選択していないときはアライメント機能を使えなくすべき
39437	2.1.1	ショートカットキー使用時に四角オブジェクトのフォーカスが表示されない
40062	2.1.1	BendpointConnectionRouter の特定のノード接続処理をうまく処理していない
45174	2.1.1	回路図の接続が同じ地点から作成した場合，回路外と接続される
78834	3	ToolEntry のサブクラスにも設定を変更できるよう許可してほしい
98345	3.1	接続レイヤーに対してのアンチエイリアス実装希望
108589	3.1	ShortestPathConnectionRouter の接続を追加したり削除したるすると NullPointException が起きる
118924	3.1	RoutingListener と PolylineConnection の追加
120077	3.2	コネクション作成に ConnectionDragCreationTool を使用するとマウスカーソルで不具合がおきる
141770	3.2	FixedConnectionAnchor の接続アンカー数が 2 個以上になるはずないのになっていしまう
143310	3.2	EditPartViewer の getFocus 関数がうまく動いていない
161040	3.2	DeferredUpdateManager の performValidation() の処理がおかしい
174085	3.2.1	既に作成されているノードと別のノードを接続するときフォーカスが解除される
175758	3.2.1	ScrollableThumbnail でのスクロールが重い
193067	3.2.1	Graphics setAlpha() がプラットフォームによって動作が異なる
212280	3.2.1	org.eclipse.draw2d.RelativeBendpoint の関数 getLocation() の戻り値が正しくない
217072	3.3.1	BendpointEditPolicy の関数 setReferencePoints() で IndexOutOfBoundsException の発生
239737	3.4	setSelection で接続が選択されない

表 5 単語“ connection ”の抽象化バグ

抽象化バグ No.1:必要な関数 (機能) が呼び出されていない，もしくは呼び出し順序がおかしい		
バグ報告 ID	バージョン	バグ要約
37145	2.1.1	接続やノードを削除したときに ArrayIndexOutOfBounds が投げられる．削除処理を二回行ってるため
37201	2.1.1	NullConnectionRouter が接続部分の位置を更新しない
37611	2.1	ソースノード削除時のフィードバック表示が削除されずに残る
39437	2.1.1	ショートカットキー使用時に四角オブジェクトのフォーカスが表示されない
抽象化バグ No.2:キーイベントが正しいキーに割り当てられていない		
バグ報告 ID	バージョン	バグ要約
33366	2.1	v2.1 で Ctrl+\ のキーイベント処理が変わったように思える．特殊なキーコードなので GraphicalViewerKeyHandler.acceptConnection へ’u001c’ を加えてください
36466	2.1.1	接点の位置調節を行った後，カンマキーに割れ当てられた機能が働かない
抽象化バグ No.3:リスナーイベントの処理順序が正しくない，もしくは呼び出されてない処理がある		
バグ報告 ID	バージョン	バグ要約
35312	2.1	マーキーツールと接続作成ツールがおかしい．デフォルトで unloadWhenFinished されるはずがない
36221	2.1	ソースオブジェクトと目的オブジェクトを繋いでソースオブジェクトを削除すると繋いだ部分のダミーが残る.EditPartListener に setconnectionSource 関数を追加し，それに関するイベントリスナーを削除した
37446	2.1	graphical viewer で接続部分を選択した後 treeviewer を選択するとマウスイベント (mousemove) がすべて treeviewer に投げられるようになる
45174	2.1.1	回路図の接続が同じ地点から作成した場合，回路外と接続される
抽象化バグ No.4:値チェック (上限，下限，真偽) が正しくない		
バグ報告 ID	バージョン	バグ要約
141770	3.2	FixedConnectionAnchor の接続アンカー数が 2 個以上になるはずないのになっていしまう
174085	3.2.1	既に作成されているノードと別のノードを接続するときフォーカスが解除される

表 6 単語“ view ”の抽象化バグ

抽象化バグ No.1:リスナーイベントの処理順序が正しくない，もしくは呼び出されてない処理がある		
バグ報告 ID	バージョン	バグ要約
35581	2.1	アウトラインで何も選択していない状態で削除コマンドが現れる
51793	2.1	プロパティでの高度な設定が設定できないにもかかわらず enable 状態になっている
69098	3	Flyout パレットが他にフォーカスがかかっているときに開くことができない
69617	3	パースペクティブの切り替えが機能しなくなる

表 7 単語“ size ”の抽象化バグ

抽象化バグ No.1:関数のパラメータと戻り値に注意する		
バグ報告 ID	バージョン	バグ要約
36451	2.1.1	ポイント配列に新しい 2 つのポイントのポインタを用意して 1 つ目にポイントをセットした後，1 つ目のポインタをリセットした後に，ポイント配列をコピーした時に ArrayIndexOutOfBoundsException が発生．動的配列の生成に渡すサイズのパラメータが間違っていた
37610	2.1	ScrollPaneSolver が間違った数字を中の図形へ渡している．数字を渡さないようにした
抽象化バグ No.2:値の最大値最小値を意識する		
バグ報告 ID	バージョン	バグ要約
37412	2.1	リサイズ可能な EditParts で左もしくは上側から小さくなるほうへリサイズしていき逆の方向へリサイズ領域を延ばしていくと最小サイズにリサイズされると同時に EditParts が動かされる．逆方向へリサイズできないように変更した
56622	2.1.1	大きいサイズのイメージを作成をする時に SWTError が出て失敗する．イメージの作成が失敗した時はダブルバッファに書き込まない
抽象化バグ No.3:関数内で必要な処理が呼び出されていない		
バグ報告 ID	バージョン	バグ要約
134163	3.1	Palette がアニメーション終了時にフラッシュする
212319	3.4	フローテキストがボーダープロパティを描かない．パッチ修正.restoreState() 関数を呼び出した
224116	3.3.1	ノードとエッジにハイライトをあてた後セクションをクリアすると IndexOutOfBoundsException を投げる．ハイライトの状態のチェック
244387	3.4	フィルター追加して削除後のグラフビューアでメモリリーク．パッチ修正．クリア処理追加

表 8 単語“ select ”の抽象化バグ

抽象化バグ No.1:リスナーイベントの処理順序が正しくない，もしくは呼び出されてない処理がある		
バグ報告 ID	バージョン	バグ要約
32847	2.1	右クリックでパレットボタンを押すとボタンがプレス状態に見えるようになる
34714	2.1	DrawEntryPage でチェックボックス作成すると起動時にチェックボックスにタブストップがかからず飛ばされる
36221	2.1	ソースオブジェクトと目的オブジェクトを繋いでソースオブジェクトを削除すると繋いだ部分のダミーが残る.EditPartListener に setconnectionSource 関数を追加し，それに関するイベントリスナーを削除した
37228	2.1	編集パーツを複数選択している時にひとつのパーツの選択を解除しようとすると直接編集モードに入ってしまう．SHIFT，ALT，CONTROL キーで直接編集モードにはいらないようにした
39437	2.1.1	ズーム後のフォーカスされている四角の図形の描画に問題がある．AncestorListener がリペイントのイベントを投げていない
抽象化バグ No.2:インターフェースの翻訳がおかしい		
バグ報告 ID	バージョン	バグ要約
70152	3	”Plugin Dependencies”の文字列だけ翻訳されておらず英語のままで，日本語になっていない
105673	3.1	メニューの文字に不具合がある

報告 ID37611,39437 について抽象化を行うと“ クリア関数 (機能) が呼び出されていない ”,“ リペイント関数 (機能) が呼び出されていない ”となる．

これら 4 つのバグ報告の抽象化に共通している部分として“ 関数 (機能) ”,“ 呼び出されていない ”,“ 二回 ”というキーワードが発見でき，この 4 つを抽象化することで“ 必要な関数 (機能) が呼び出されていない，もしくは呼び出し順序がおかしい ”という抽象化バグが作成できる．“ size ”,“ connection ”,“ select ”,“ view ”の 4 つの各単語で抽出したバグ報告の ver2.x に該当するバグ報告でバグの抽象化を行った．

4.2 抽象化バグからチェックリスト作成

4.1 で作成した ver2.x の抽象化バグをチェックリストの項目とした結果，4 つの単語を含むバグ報告内で該当するバグ件数は表 9 のようになった．抽象化バグを作成しても該当しない抽象化バグがあることが分かる．

5. 考　　察

バグ報告に含まれるバグの症状，再現方法，バグの原因の記述をもとにバグ要約を作成した．バグ要約から，そのバグの全体像を得た．また，バグ報告に含まれる単語を抽出，計数

表 9 四単語より作成した抽象化バグに該当する ver3.x バグ報告件数

抽象化バグ No.1:必要な関数 (機能) が呼び出されていない，もしくは呼び出し順序がおかしい	
抽出単語	件数
connection	4
select	1
抽象化バグ No.2:キーイベントが正しいキーに割り当てられていない	
抽出単語	件数
該当なし	– –
抽象化バグ No.3:リスナーイベントの処理順序が正しくない，もしくは呼び出されていない処理がある	
抽出単語	件数
connection	1
view	2
select	1
抽象化バグ No.4:関数のパラメータと戻り値に注意する	
抽出単語	件数
view	1
size	1
抽象化バグ No.5:値チェック (上限、下限、真偽) が正しくない	
抽出単語	件数
size	1
connection	2

することにより複数のバグ要約をもとに抽象化バグを得ることができた．バグ要約をもとにチェックリストを作成した場合に，どの程度のバグを未然に防ぐことができるかを調べたところ，5 つの抽象化バグに該当する 14 件のバグ報告をみつけることができた．

試行では，出現頻度別単語一覧を作成する際に GUI アプリケーションに関連する単語に注目して選別を行い，抽象化の際にそれらの単語が残るように (GUI アプリケーションに関係する単語は抽象化の対象にしないように) した．そのため “キーイベント”，“リスナーイベント” といった単語を残したまま抽象化ができ，該当するバグをみつけられる可能性が高まった．

しかしながら，GUI アプリケーションに関連する単語が含まれていない場合には，抽象的すぎて，不具合が発見できない可能性のある抽象化バグとなったものもある．文献[9] でも指摘されているが，抽象度の高すぎるチェックリストはバグ発見に役立たない可能性が高い．たとえば，表 9 の No. 4: “関数のパラメータと戻り値に注意する” といった抽象化バグでは，該当するバグが存在する場合でも，発見できない可能性がある．抽象化について，GUI アプリケーションに関連する単語以外の何らかの制約が必要であると考える．

また，抽出した抽象化バグがチェックリストの項目になりえるか確認するため，2.x で作成した抽象化バグに該当する 3.x のバグ報告があるか試みた．その結果，調査した 86 件のバグ報告の中で 14 件のバグが未然に防げた可能性があることを確認した．

6. おわりに

複数のバグ報告をもとにバグを抽象化するための手順を提案した．提案した手順では，既に解決したバグ報告を対象として，バグの症状と原因を説明する文章に現れる単語とその出現頻度を計数する．計数した単語から前置詞等の一般的な単語を削除した上で，出現頻度の高い単語を含むバグ報告から，バグの症状を抽象化する．単語の出現頻度を用いることにより，客観的かつ網羅的なバグの抽象化ができる．

提案した手順をオープンソースソフトウェアのバグ管理システムに適用した．対象ソフトウェアは，Eclipse のプラグインの 1 つである GEF というソフトウェアである．GEF のバグ管理システムには，400 件超の解決済みバグが登録されている．最新バージョンは 3.5 である．バグ管理システムに登録されている全てのバグ報告から単語の出現頻度を計数し，その単語をもとにバグを抽象化した．また，4 つの単語を選び，2.x で発見されたバグを抽象化し，チェックリスト化した．また，3.x で報告されているバグをチェックリストにより未然に防げていたかどうかを確認したところ，未然に防げていた可能性のあるバグを確認できた．

今後の課題として，バグ抽象化手順の自動化支援，他言語での試行，抽象化コストの計測と未然にバグを防げた場合に省略できる修正コストを比較することが挙げられる．

謝　辞

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」の委託に基づいて行われた．

参考文献

1) 野中 誠, “ 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 ”, 情報処理学会研究報告, VOL.118, pp.25-31, 2004.
2) Per, Runeson., Magnus, Alexandersson., Osker, Nyholm., “ Detection of Duplicate Defect Reports Using Natural Language Processing ”, Proceedings of the 29th international conference on Software Engineering, pp.499-510, 2007.
3) Chernak, Y., “ A statistical approach to the inspection checklist formal synthesis and improvement ”, IEEE Transactions on Software Engineering, VOL.22, pp.866-874, 1996.
4) Thelin, T., Runeson, P. and Regnell, B., “ Usage-Based Reading – An Experiment to Guide Reviewers with Use Cases ”, Information and Software Technology, VOL.43, pp.925-938, 2001.
5) Adam A. Porter., Lawrence G. Votta, Jr., Victor R. Basili., “ Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment ”, IEEE Transactions on Software Engineering, VOL.21, pp.563-575, 1995.
6) Victor R. Basili., Scott Green., Oliver Laitenberger., Forrest Shull., Sivert Sorumgard., Marvin V. Zelkowitz., “ The empirical investigation of perspective-based reading ”, University of Maryland at College Park, pp.41, 1995.
7) Thomas, Thelin., Carina, Andersson., Per, Runeson., Nina, Dzamashvili-Fogelstrom.,“ A Replicated Experiment of Usage-Based and Checklist-Based Reading ”, Proceedings of the Software Metrics, 10th International Symposium, pp.246-256, 2004.
8) Thomas, Thelin., Per, Runeson., Claes, Wohlin., “ An Experimental Comparison of Usage-Based and Checklist-Based Reading ”, IEEE Transactions on Software Engineering, VOL.29, pp.687-704, 2003.
9) Bill, Brykczynski., “ A Survey of Software Inspection Checklists ”, Software Engineering Notes, VOL.24, No.1, pp.82-89, 1999.