Vol. 44 No. 6 情報処理学会論文誌 June 2003

ソフトウェアメトリクスを利用したリファクタリングの自動化支援機構

秦 野 克 彦[†] 乃 村 能 成[††]
谷 口 秀 夫[††,☆] 牛 島 和 夫[†††]

ソフトウェアは要求に合わせて機能変更や機能拡張され，改版されていく．この結果，設計当初のプログラム構造の統一性は崩れることが多く，機能変更や機能拡張の工数も増加してしまう．それにともない，ソフトウェアの保守の工数も増加してしまう．したがって，プログラムの構造を見直し，ソフトウェアの機能変更や機能拡張の工数を少なくすることが重要である．このためにリファクタリングが有効である．しかし，リファクタリングを行うためには，機能変更や機能拡張の工数の増加を招くプログラムの構造的欠陥を検出する必要がある．さらに，検出した構造的欠陥を解消する適切なリファクタリング手法を選択し施す必要がある．従来，こうした検出や選択は難しく，リファクタリングに関する知識や経験を必要とした．本論文では，過去の知識や経験が少なくてもプログラム構造を改善可能にするため，リファクタリング作業の自動化を支援する機構を提案する．

A Mechanism to Support Automated Refactoring Process Using Software Metrics

Katsuhiko Hatano,[†] Yoshinari Nomura,[††] Hideo Taniguchi[††,☆]
and Kazuo Ushijima[†††]

Software is continuously changed and evolved at users' requests. Consequently, software tends to lose its original concrete structure and become complex. Maintaining such software is extremely costly. Programmer needs to restructure the software before losing control of it. We can use a technique called “Refactoring”. Refactoring is to reorganize a program without changing its function. In a refactoring procedure, it is difficult for a novice programmer to find out where he applys refactoring to and which refactoring method he should use. In this paper, we propose a mechanism for supporting automated refactoring process.

1. はじめに

要求に合わせて機能変更や機能拡張を行い，ソフトウェアは改版されていく．ソフトウェア開発や保守にかかる費用の半分以上が，機能変更と機能拡張への対応に充てられており[1)]，ソフトウェアの開発や保守にかかる費用を削減するためにソフトウェアの機能変更や機能拡張の工数を少なくすることは非常に重要である．ところが，ソフトウェアを繰り返し改版していくと，設計当初の方針に基づいて作成されたプログラム構造は崩れていくことが多い[2)]．このため，機能変更や機能拡張の工数が増加してしまう．

そこで，ソフトウェア構造を見直すためにリファクタリングが有効である．リファクタリングとは，ソフトウェアが提供する機能を変更することなく，プログラムの内部構造を変更することである．このため，既存のプログラムにリファクタリングをうまく適用して，機能変更や機能拡張の工数を少なくすることができる．

しかし，リファクタリングを行うためには機能変更や機能拡張の工数の増加を招くプログラムの構造的欠陥を検出する必要がある．さらに，検出した構造的欠陥を解消する適切なリファクタリング手法を選択し施す必要がある．従来，こうした検出や選択は難しく，リファクタリングに関する知識や経験を必要とした．

そこで，本論文では，過去の知識や経験が少なくてもリファクタリング作業を行える機構を提案する．具

† 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
†† 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University
††† 九州産業大学情報科学部
Faculty of Information Science, Kyushu Sangyo University
☆ 現在，岡山大学工学部
Presently with Faculty of Engineering, Okayama University

体的には，ソフトウェアメトリクスを利用して構造的欠陥を検出することを支援し，ソフトウェアメトリクスとリファクタリング各手法の相関を明らかにして構造的欠陥を解消する適切なリファクタリング手法を選択することを支援する．これらにより，作業者の知識や経験に大きく左右されることなくプログラムを改善することができる．

2. 従来のリファクタリングと問題点

従来から，リファクタリングを利用して，機能変更や機能拡張の工数削減を目指すことが行われている．この様子を**図 1** に示す．図 1 は，既存プログラムをリファクタリングにより再構成する様子を示している．図 1 において，不具合の兆候とは，機能変更や機能拡張の工数の増加となるようなプログラムの構造的欠陥であり，何らかの原因をともなう．たとえば，巨大なクラスが存在するという不具合の兆候には，クラスの責務が大きすぎるという原因がある．Fowler は，不具合の兆候を「不吉な匂い」と呼び，22 個を示している[2]．リファクタリング手法とは，プログラムを再構成する操作である．たとえば，メソッドの移動やクラスの抽出といったものがある．ソフトウェアメトリクスとは，ソフトウェアの規模や複雑さ，機能性，構造といった観点からソフトウェアを数値化できる尺度であり[3]，代表的なものとして，C&K メトリクスがある[4]．

以降に，図 1 に基づき，リファクタリングを利用した従来の機構における作業手順を説明する．

(1)　既存のプログラムの構造を分析し，不具合の兆候を検出する．

(2)　不具合の兆候を生み出す原因を検討し，不具合の兆候を解消するリファクタリング手法を選択する．

(3)　リファクタリング手法を既存プログラムに適用し，プログラムを再構成する．

(4)　再構成されたプログラムをソフトウェアメトリクスにより数値化する．

(5)　ソフトウェアメトリクスの数値により，リファクタリングがソフトウェアの機能変更や機能拡張の工数を少なくすることを確認する．

上記の手順において，手順 (1)，(2) が適切に行われるか否かは，作業者の知識や経験に大きく依存する．このため，手順 (5) において，リファクタリングがうまくできたと確認できるとは限らない．つまり，従来のリファクタリング機構は，作業者の知識や経験に大きく依存するため，以下の問題がある．

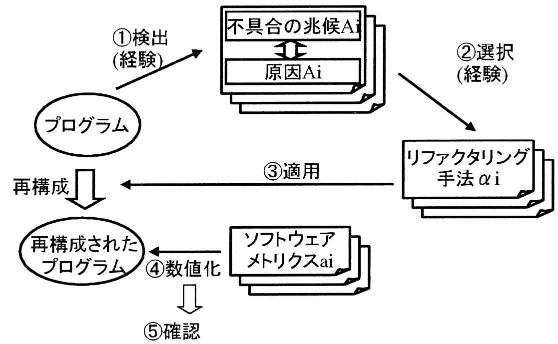


図 1　従来のリファクタリング機構
Fig. 1　The conventional program refactoring.

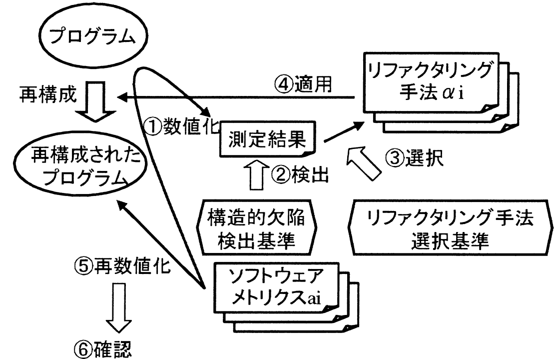


図 2　ソフトウェアメトリクスを利用したリファクタリング機構
Fig. 2　A mechanism to support program refactoring using software metrics.

(1)　不具合の兆候をうまく検出することは難しい．

(2)　不具合の兆候に対し，適切なリファクタリング手法を選択することは難しい．

また，手順 (1)，(2) において，ソフトウェアメトリクスを利用していないにもかかわらず手順 (5) でソフトウェアメトリクスを利用しているため，効果の尺度が統一できていない．

もちろん，上記の手順は一例であり，手順 (4)，(5) のように必ずソフトウェアメトリクスを用いてリファクタリングの効果を測定するとは限らない．リファクタリング開始前に設定した目的に合わせて，経験的に判断することも考えられる．

3. ソフトウェアメトリクスを利用したリファクタリング機構

3.1 基 本 機 構

従来のリファクタリングの問題点を解決するため，ソフトウェアメトリクスを利用したリファクタリング機構を提案する．提案する機構を**図 2** に示す．図 2 では,「構造的欠陥検出基準」と「リファクタリング手法

選択基準」という 2 つの知識を用意する.「構造的欠陥検出基準」とは，ソフトウェアメトリクスの測定値をもとに，当該プログラムの改善の要否を判定する基準である.「リファクタリング手法選択基準」とは，リファクタリング手法とソフトウェアメトリクスの値との増減関係である．つまり，リファクタリング手法をプログラムに適用した結果，ソフトウェアメトリクスによる測定値は，減少する，増加する，変わらない，のいずれかであり，この関係を示すものである．以降に，図 2 に基づき，機構の作業手順を説明する．

(1) ソフトウェアメトリクスにより既存プログラムを数値化し，測定結果を得る．
(2) 「構造的欠陥検出基準」を利用し，測定結果からプログラムの構造的欠陥を検出する．
(3) 「リファクタリング手法選択基準」を利用し，構造的欠陥を解消するリファクタリング手法を選択する．
(4) リファクタリング手法を既存プログラムに適用し，プログラムを再構成する．
(5) 再構成されたプログラムをソフトウェアメトリクスにより数値化する．
(6) ソフトウェアメトリクスの数値の変化により，リファクタリングによって構造的欠陥を解消できたことを確認する．

以上のことから，作業者の知識や経験に頼ることなくリファクタリングを行える．具体的には,「構造的欠陥検出基準」を用いることで，不具合の兆候を生む構造的欠陥を検出でき,「リファクタリング手法選択基準」を用いることでリファクタリング手法を選択できる．また，ソフトウェアメトリクスを作業手順の最初と最後で利用することにより，プログラムの構造的欠陥を統一的に定量化でき，リファクタリングの効果を把握しやすくなっている．したがって，本機構により，先に示した従来の機構の問題を解消できる．

3.2 構 成 要 素

提案機構を実現するために必要な構成要素について述べる．構成要素とは，図 2 に示したソフトウェアメトリクス，リファクタリング手法，構造的欠陥検出基準，およびリファクタリング手法選択基準である．

3.2.1 ソフトウェアメトリクス

ソフトウェアメトリクスには様々なものがあり，特にオブジェクト指向言語に関するものは，オブジェクト指向ソフトウェアメトリクスと呼ばれている．オブジェクト指向ソフトウェアメトリクスは，オブジェクト指向言語に特有な属性（継承，カプセル化，多態性）を表す尺度になっている．代表的なものに C&K メトリクス[4]，青木メトリクス[5]，Lorenz&Kidds Metrics[6]，MOOD[7]がある．

青木メトリクスは，プログラム全体から見たクラスの役割を測定するための尺度である．たとえば，クラスは抽象度と具体度のどちらが高いか，クラスは集約をしていて全体度が高いか集約されていて部分度が高いかといった尺度である．このため，こうした観点を持つソフトウェアメトリクスは，プログラムの構造的欠陥を見つけるには不向きである．また，MOOD も同様の尺度である．一方で，C&K メトリクスはクラスに対する複雑さを測定するための尺度である．たとえば，クラスの大きさ，クラスの継承の深さ，およびクラス間の結合度といったものを表す．C&K メトリクスによる測定値は，値が大きなほど，改善を必要とすることを示す．また，Lorenz&Kidds Metrics も同様の観点の尺度を持つ．つまり，この観点のソフトウェアメトリクスを利用することで，プログラムの構造的欠陥を測定値の大小で判断できるといえる．

しかしながら，これらのソフトウェアメトリクスはリファクタリングを支援するために開発されたものではない．そのため，リファクタリング支援の観点から必要なプログラムの評価尺度を十分に提供しているとはいえない．そこで，提案する本機構では，利用するソフトウェアメトリクスを容易に変更可能で，複数のソフトウェアメトリクスを同時に利用可能な構成にしている．そのために，ソフトウェアメトリクスとリファクタリングの関係をリファクタリング手法選択基準として定義可能にしている．

3.2.2 リファクタリング手法

リファクタリング手法は，プログラムを再構成する操作である．代表的なリファクタリング手法として，Opdyke によるもの[8]と Fowler によるもの[2]がある．

Opdyke は，形式的な側面からリファクタリング手法について議論をしている．具体的には次の 2 点である．

(A) リファクタリング手法を施す際にプログラムが満たしておくべき前提条件
(B) 各リファクタリング手法がプログラムの振舞いを保つことの証明

Opdyke は 29 個のリファクタリング手法を 26 個の原始的な低レベルリファクタリング手法と 3 個の高レベルリファクタリング手法に分類している．高レベルリファクタリング手法は，低レベルリファクタリング手法を組み合わせて行う．このため，上記 (A)，(B) に加え，さらに，

(C) 低レベルリファクタリング手法を施す手順

が議論されている．

Fowler は，リファクタリング手法をプログラムを再構成する行為のパターンととらえている．そして，リファクタリングのノウハウを提供する目的で，72 個の代表的なリファクタリング手法を次の形で記述しカタログ化している．

- 名前：リファクタリング手法の名称
- 要約：リファクタリング手法を必要とする状況とリファクタリング手法を施す手順の概要
- 動機：リファクタリング手法を必要とする理由
- 手順：リファクタリング手法を施す手順
- 例：手順に沿った適用例

なお，Fowler のリファクタリング手法を組み合わせて，Opdyke の高レベルリファクタリング手法を施すことができる．

3.2.3 構造的欠陥検出基準

構造的欠陥検出基準は，ソフトウェアメトリクスの測定値をもとに当該プログラムの改善の要否を判定する基準である．具体的には，ソフトウェアメトリクスの測定値の大小がプログラム構造の悪い良いに対応するため,「測定値の偏差値」を用いてプログラムの改善の要否を判定できる．これは,「測定値の偏差値」が構造的欠陥の程度を表すからである．

3.2.4 リファクタリング手法選択基準

リファクタリング手法選択基準とは，リファクタリング手法とソフトウェアメトリクスの値の増減関係である．具体的には，リファクタリング手法 α_i をプログラム P_i に適用したとき，ソフトウェアメトリクス a_i の値は，減少，増加，不変のいずれかである．この測定値の増減関係がリファクタリング手法選択基準である．これにより，ソフトウェアメトリクス a_i の測定値を効果的に増加あるいは減少させたいリファクタリング手法 α_i を選択できる．

この基準をすべてのプログラムとリファクタリング手法について明らかにすることは困難である．このため，ソフトウェアメトリクスの観点とリファクタリング手法の内容からこの基準を明らかにする必要がある．

4. 実現機構と適用例

4.1 実現機構

ここでは提案機構の 4 つの要素を具体化する．なお，ここでは，対象プログラミング言語をオブジェクト指向言語である ObjectPascal 言語とした．

4.1.1 ソフトウェアメトリクス

提案した機構には，リファクタリングを指向したソフトウェアメトリクスが必要である．しかし，ソフトウェアメトリクスの確立には多大な工数が必要である．そこで，ソフトウェアの複雑度を示す既存のソフトウェアメトリクスとして，オブジェクト指向ソフトウェアメトリクスである C&K メトリクスを採用した．C&K メトリクスは，オブジェクト指向ソフトウェアメトリクスのデファクト標準だからである[9]．

C&K メトリクスは，複雑度を表す 6 つの尺度からなる．これら 6 つの尺度はクラスを測定単位としているので，1 つのクラスに対して 6 つの測定値が定まる．測定値は 0 以上の整数値をとり，値が大きなほどクラスが複雑であることを意味する．C&K メトリクスの各尺度を簡単に以下に説明する．

WMC Weighted Methods per Class：あるクラスの重み付きメソッド数

DIT Depth of Inheritance Tree：あるクラスの継承木における深さ

NOC Number Of Children：あるクラスの持つ直接のサブクラスの数

CBO Coupling Between Object classes：あるクラスがメソッドを介して参照しているクラスの数

RFC Response For a Class：あるクラスのオブジェクトがメッセージを受信したとき，それに反応して実行されうるメソッド数

LCOM Lack of COhesion in Methods：あるクラスのカプセル化の度合い．インスタンス変数を共有しないメソッドの組の数からインスタンス変数を共有するメソッドの組の数を引いた数（計算結果が負のときは 0 と見なされる）

4.1.2 リファクタリング手法

実現機構のリファクタリング手法として，Fowler による 72 個のリファクタリング手法を用いた．なぜならば，Fowler のリファクタリング手法は，Opdyke のものよりも数が多く，Fowler のリファクタリング手法を組み合わせることで Opdyke のリファクタリング手法を施すことができるからである．

4.1.3 構造的欠陥検出基準

構造的欠陥検出基準として偏差値を用いた．クラスが複雑なほど，C&K メトリクスによる測定値は，大きくなり，偏差値は高くなる．そこで，構造的欠陥の程度を偏差値で表す．偏差値を用いることで，異なる尺度の間でもその尺度の表す構造的欠陥の程度を共通に比較することができる．

偏差値を求めるために，平均と標準偏差を与える母集団として，インターネット上でプログラムのソースコードが公開されている ObjectPascal 言語で記述されたプログラムを 15 個収集した．プログラムの具体

表 1 Object Pascalプログラムに対するC&Kメトリクスの平均値および標準偏差値
Table 1 The values of mean and standard deviation by C&K Metrics for Object Pascal programs.

尺度	WMC	DIT	NOC	CBO	RFC	LCOM
平均	8.4	4.0	0.9	4.4	45.7	48.1
標準偏差	11.2	2.8	3.2	6.1	75.6	267.8

的な内訳は，ファイル解凍/圧縮プログラム3，ブラウザプログラム1，ゲームプログラム3，文字列操作プログラム3，ネットワーク通信プログラム5の計15個である．15個のプログラムの総コード行数は52695行，総クラス数は190個である．

収集したプログラムに対する測定結果からC&Kメトリクスの尺度別に平均と標準偏差を求めた．結果を**表 1** にまとめる．なお，測定にあたって尺度WMCの重みを1とした．つまり，尺度WMCの測定値は，被測定クラスで定義されたメソッド数を示す．

4.1.4 リファクタリング手法選択基準

リファクタリング手法選択基準を用いて構造的欠陥を解消するリファクタリング手法を選択する．Fowlerのリファクタリング手法について，各々のリファクタリング手法の適用によって，C&Kメトリクスの各尺度の値がどのように変化するか調べた．結果を**表 2** に示す．なお，Fowlerのリファクタリング手法72個のうち，リファクタリング手法を適用しても，C&Kメトリクスの6つの尺度すべてについて，測定値が変化しないものは30個あった．このため，それら30個を除いた42個について表2に示している．表2において，各欄に入る記号は■は必ず減少，●は減少する可能性あり，△は不明，○は増加する可能性あり，および□は必ず増加を示す．空欄は，尺度の値が変化しないことを意味する．

表2に示す結果を得る方法について例をあげて説明する．**図 3** は,「メソッドの移動」について説明した図である．Class1の持つメソッドaMethodをClass2に移動している．移動元のClass1について，C&Kメトリクスの各尺度の値の変化を考える．まず，Class1のメソッドの数は減少するので，WMCは減少する．Class1が移動先のメソッドを参照する委譲メソッドを用意する可能性を考慮して，WMCは減少する可能性あり（●）とした．CBOについても，aMethodを呼び出していた他のクラスとの関係が減少する可能性があるため，●とした．また，メソッドが減少することにともなって，Class1のオブジェクトがそのメソッド中から発行していたメソッド呼び出しがなくなる．そのため，Class1において潜在的に実行されうるメソッドの数が減少することになる．したがって，RFCについても，減少する可能性あり（●）とした．メソッドの移動は，クラス階層を変更することはないので，DITおよびNOCは不変とした．

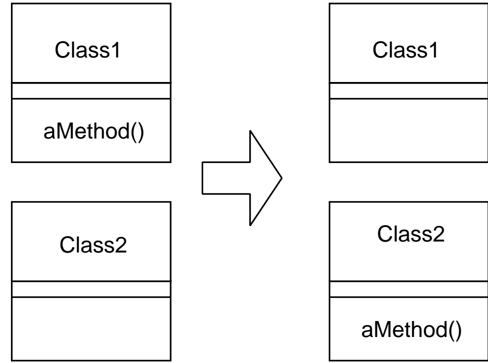


図 3 メソッドの移動
Fig. 3 Move method.

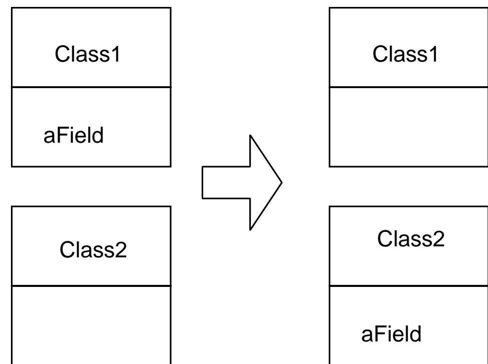


図 4 フィールドの移動
Fig. 4 Move field.

図 4 は,「フィールドの移動」について説明した図である．Class1の持つフィールドaFieldをClass2に移動している．移動元のClass1について，C&Kメトリクスの各尺度の値の変化を考える．まず，Class1では，aFieldが減少することによって，直接的にはメソッドの増減はない，しかし，aFieldを参照している部分が多い場合は，それらのために，アクセサgetField, setFieldを定義することがある．したがって，Class1におけるメソッドの数WMCは増加する可能性がある（○）．getField，setFieldはClass2に移動されたaFieldを参照することになるので，Class1において潜在的に実行されるメソッドの数RFCも増加する可能性がある（○）．また，フィールドが減少することによって，クラスの凝集性が高まるため，LCOMは減少する可能性（●）がある．

表 2　Fowler のリファクタリング手法と C&K メトリクスの各尺度との関係
Table 2　The relationships between Fowler's refactoring methods and C&K Metrics.

リファクタリング手法	尺度					
	WMC	DIT	NOC	CBO	RFC	LCOM
メソッドの移動	●			●	●	△
フィールドの移動	○				○	●
クラスの抽出	□			●	●	
クラスのインライン化	□			■	□	□
委譲の隠蔽	□				□	□
仲介人の除去	●				●	●
外部メソッドの導入	□				□	△
局所的拡張の導入	■			△	■	■
フィールドの引き上げ	□			○	□	△
メソッドの引き上げ	□			○	□	△
コンストラクタ本体の引き上げ	□			○	□	△
フィールドの引き下げ	■			●	■	△
メソッドの引き下げ	■			●	■	△
サブクラスの抽出	■		□	●	■	■
スーパークラスの抽出	■	□		●	■	■
インタフェースの抽出				□		
階層の平坦化	□	●	●	○	□	△
TemplateMethod の形成	□			○	□	△
委譲による継承の置き換え	□	■		□	□	△
継承による委譲の置き換え	■	□		■	■	△
メソッドの抽出	□				△	△
問い合わせによる一時変数の置き換え	□				□	△
メソッドオブジェクトによるメソッドの置き換え	■			□	□	△
アルゴリズムの取り替え					△	
条件記述の分解	□				□	△
条件記述の統合	□				□	△
ポリモーフィズムによる条件記述の置き換え	●				●	●
ヌルオブジェクトの導入				□	■	△
表明の導入				○	○	
問い合わせと更新の分離	□				□	△
メソッドのパラメータ化	△				△	△
明示的なメソッド群による引数の置き換え	□				□	△
オブジェクトそのものの受渡し					●	△
引数オブジェクトの導入				□	□	
set メソッドの削除	■					□
Factory Method によるコンストラクタの置き換え	□				□	△
自己カプセル化フィールド	□				□	■
オブジェクトによる配列の置き換え				□	□	△
観察されるデータの複製				□	□	△
フィールドのカプセル化	□				□	■
コレクションのカプセル化	□				□	■
フィールドによるサブクラスの置き換え	□		■		□	△

■：必ず減少，●：減少する可能性あり，△：不明，
○：増加する可能性あり，□：必ず増加，空欄：変化なし

4.2　適　用　例

本節では，実プログラムに対してリファクタリングを行った内容について述べる．リファクタリングの対象である実プログラムは，オブジェクト指向プログラミング言語である ObjectPascal 言語で書かれており，C&K メトリクスに基づいて入力プログラムの複雑さを測定するもので，ソースコードを与えると画面上に測定結果を出力するものである．この対象プログラムの規模は総行数で 9,543 行であり，新規に作成された 12 個のクラスと再利用された 4 個のクラスからなる．以降，新規に作成された 12 個のクラスをリファクタリングの対象とし，図 2 に示した機構の各手順に従い，各項で内容を述べる．

4.2.1　数値化と検出

対象プログラムを C&K メトリクスで数値化し，測定結果を得る．**表 3** は対象プログラムの新規に作成さ

表 3 対象プログラムの偏差値
Table 3 The deviation values of a sample program.

クラス名	WMC	DIT	NOC	CBO	RFC	LCOM
THash	52.3	49.9	50.2	46.0	47.7	48.2
EnScope	45.2	53.4	59.5	44.4	45.0	48.2
EnApplication	44.3	57.0	47.2	44.4	44.5	48.2
EnUnit	44.3	57.0	47.2	44.4	44.5	48.2
EnClass	49.6	57.0	47.2	47.7	48.6	49.2
EnMethod	44.3	57.0	47.2	46.0	45.0	48.2
TForm1	62.2	64.1	47.2	69.0	58.5	55.3
TForm2	42.5	64.1	47.2	42.7	43.9	48.2
TEnv	65.7	49.9	47.2	64.1	66.2	50.7
TCustomParser	50.5	42.7	53.3	50.9	52.4	49.0
TUnitParser	62.2	46.3	47.2	49.3	55.3	56.5
TProgramParser	44.3	46.3	47.2	46.0	44.5	48.2

れた 12 個のクラスを C&K メトリクスで測定した結果であり，偏差値を示す．下線は偏差値が最大のものである．

次に，対象プログラムから構造的欠陥を検出する．偏差値の大小は構造的欠陥の程度を表しており，偏差値が 50 より大きいほどプログラムの構造的欠陥の程度が大きいといえる．したがって，ここでは偏差値の最大値を持つ TForm1 の CBO を最初に，偏差値の高い順に各クラスへリファクタリング手法を適用する．

4.2.2 選択と適用と再数値化

構造的欠陥を解消するリファクタリング手法を選択する．ここでは，偏差値の最大値を持つ TForm1 の CBO を対象とする．尺度 CBO の測定値を下げるリファクタリング手法は表 2 によれば 8 つある．これらが適用するリファクタリング手法の候補である．

ここで，どの手法を適用するか決定するために，リファクタリング手法の効果の度合いを計算した．具体的には，表 2 における測定値を増減させる記号（■，●，△，○，□）にそれぞれ数値（2，1，0，−1，−2）を割り当て，表 2 における空欄には数値 0 を割り当てた．そして，それら数値の総和を効果の度合いとした．たとえば，表の 1 番上に現れるリファクタリング手法「メソッドの移動」は，記号●を 3 つと記号△を 1 つ持つ．したがって，効果の度合いとして値 3 を持つ．尺度 CBO の測定値を下げるリファクタリング手法とその効果の度合いを**表 4** に示す．表 4 では，効果の度合いが大きな順にリファクタリング手法を並べ直している．

次に，効果の度合いが大きい順に適用するリファクタリング手法を検討する．検討の結果「クラスの抽出」を適用することにした．検討内容を記すと，まず，「フィールドの引き下げ」と「メソッドの引き下げ」は，引き下げ先のサブクラスを必要とする．TForm1 はサブクラスを持っていない（NOC の値=0）ので適用できない．また,「継承による委譲の置き換え」は委譲メソッドがたくさんある場合，継承を使って委譲メソッドを置き換えるものである．TForm1 は委譲メソッドを持っていないので適用できない．さらに,「サブクラスの抽出」は，特定のインスタンスだけに必要な特性を抽出するものである．TForm1 はインスタンスを 1 つしか持たないので適用できない．そして,「スーパークラスの抽出」は 2 つ以上の類似するクラスから共通部分をスーパークラスとして抽出するものである．TForm1 は抽出するべき共通する部分を持っていないので適用できない．最後に,「メソッドの移動」は，参照先のクラスにメソッドを移動するものである．TForm1 は参照先のクラスがすべてクラスライブラリであるため，適用できない．以上により「クラスの抽出」の適用を決定した．

表 4 尺度 CBO の測定値を下げるリファクタリング手法とその効果の度合い
Table 4 The refactoring methods decreasing the measured value of CBO and their effectiveness.

リファクタリング手法	効果の度合い
フィールドの引き下げ	5
メソッドの引き下げ	5
継承による委譲の置き換え	4
サブクラスの抽出	4
スーパークラスの抽出	4
メソッドの移動	3
クラスの抽出	0
クラスのインライン化	−4

その次に，リファクタリング手法を適用する.「クラスの抽出」はクラスを新たに作って，フィールドとメソッドを元のクラスからそこへ移動するリファクタリング手法である．この場合は，新たに 4 つのクラスを作って TForm1 のフィールドとメソッドを各クラスに移動した．

最後に，再構成されたプログラムを C&K メトリクスで測定し再び数値化する．

リファクタリング手法の適用により各尺度の測定値は増減し，それに合わせて偏差値も増減する．そのため，リファクタリング手法を施すたびに対処する順序も変わっていく．対処順序が変わっても，その時点で最大値となる偏差値に注目して，引き続きリファクタリング手法の選択，適用，再数値化を繰り返した．具体的には，TEnv の RFC について「サブクラスの抽出」，TUnitParser の WMC について「サブクラスの抽出」，そして EnScope の NOC について「階層の平坦化」を適用した.「サブクラスの抽出」により，TEnv のサブクラス TAppEnv と TUnitParser のサブクラ

表 5 リファクタリングによる偏差値の変化
Table 5 The change of deviation values before and after program refactoring.

クラス名	WMC	DIT	NOC	CBO	RFC	LCOM	偏差値の平均
TForm1	62.2 ↘ 52.3	64.1	47.2	69.0 ↘ 55.9	58.5 ↘ 48.8	55.3 ↘ 48.5	59.4 ↘ 52.8
TEnv	65.7 ↘ 55.0	49.9	47.2 ↗ 50.2	60.8 ↘ 59.1	66.2 ↘ 58.5	50.7 ↘ 49.1	56.7 ↘ 53.7
TUnitParser	62.2 ↘ 45.2	46.3	47.2 ↗ 50.2	49.3 ↘ 47.7	55.3 ↘ 47.5	56.5 ↘ 48.3	52.8 ↘ 47.5
EnScope	45.2 ↗ 50.5	53.4	59.5 ↘ 47.2	44.4 ↗ 46.0	45.0 ↗ 49.6	48.2 ↗ 49.3	49.3 ↗ 49.4

ス TCustomUnitParser が抽出された．また,「階層の平坦化」により，EnScope のサブクラスである 4 つのクラス（EnApplication，EnUnit，EnClass および EnMethod）は EnScope へ統合された．なお，途中過程で TForm1 の DIT と TForm2 の DIT について改善対象となったが，これらには適用可能なリファクタリング手法がなかったため，改善対象から除いた．

4.2.3 確 認

リファクタリング手法の適用による偏差値の変化から，構造的欠陥を解消できたことを確認する．一連のリファクタリング手法の適用による偏差値の変化を**表 5** に示す．多くの場合，偏差値の算出には測定値自身を母集団に含める．しかしながら，リファクタリング手法の適用後，プログラムの構造が変化するため，測定して得られる測定値も変化する．そのため，測定値自身を母集団に含めて偏差値を算出すると，リファクタリング手法の適用前後で偏差値の算出基準である平均や標準偏差も変化してしまう．偏差値の変化を調べるうえで，リファクタリング手法の適用前後で算出基準の異なる偏差値を比較することは適切でないと考えた．したがって，ここではつねに表 1 の値を基準として偏差値を算出している．偏差値の変化において，増加は ↗，減少は ↘ で表した．また，下線は解消しようとした構造的欠陥を示す偏差値である．4 つのクラスについて下線の偏差値は最大で 27.3%，最低でも 11.6%下がっており，構造的欠陥を解消したことを確認できた．また，各クラスに対するリファクタリング手法の適用による改善効果を評価するために，各クラスに対する偏差値の平均値の変化を示す．TForm1, TEnv，および TUnitParser については，偏差値の平均が減少しており，改善効果が見られる．EnScope については若干ながら偏差値の平均が増加している．しかし，NOC の値を 12.5 も低下させ，かつ他の尺度は増加したもののその値はせいぜい 50 程度であり，改善効果があったと判断できる．

全体の結果を**表 6** に示す．表 6 の下線は偏差値が最大のものである．リファクタリング手法を適用できなかった TForm1 の DIT と TForm2 の DIT が偏差値の最大値 64.1 をとっている．それ以外では，偏差値 60 を超える値はなく，全体として構造的欠陥は解消されている．

表 6 対象プログラムの偏差値（最終状態）
Table 6 The deviation values of the sample program (final state).

クラス名	WMC	DIT	NOC	CBO	RFC	LCOM
THash	52.3	49.9	50.2	46.0	47.7	48.2
EnScope	50.5	53.4	47.2	46.0	49.6	49.3
TProgramParser	44.3	46.3	47.2	46.0	44.5	48.2
TForm1	52.3	64.1	47.2	55.9	48.8	48.5
TUnitsTreeView	47.8	39.2	47.2	50.9	49.0	48.5
TAllUnitsTreeView	47.0	39.2	47.2	47.7	45.8	48.4
TMetricsListView	47.0	39.2	47.2	52.6	50.0	48.2
TmainForm	47.0	39.2	47.2	55.9	52.7	48.2
TForm2	42.5	64.1	47.2	42.7	43.9	48.2
TEnv	55.0	49.9	50.2	52.6	55.9	49.1
TAppEnv	54.1	53.4	47.2	50.9	51.5	49.2
TCustomParser	50.5	42.7	53.3	50.9	52.4	49.1
TUnitParser	45.2	46.3	50.2	47.7	47.5	48.3
TCustomUnitParser	57.7	49.9	50.2	49.3	51.5	53.2
TFullUnitParser	44.3	53.4	47.2	42.7	44.2	48.2

5. 関 連 研 究

リファクタリングに基づく eXtreme Programming と呼ばれるソフトウェア開発手法が提唱されている[10]．この開発手法では，リファクタリングに関する十分な知識や経験を必要としている．

リファクタリングはプログラムの構造的欠陥を検出し適切なリファクタリング手法を選択するという過程と，実際にリファクタリング手法を適用する過程からなる．初期の研究では後者の過程に焦点があてられており[8),11)]，リファクタリング手法を適用する過程を自動化するツールついての研究がなされている[12]．これに対し本論文は，リファクタリング手法を適用するまでの過程に注目し，知識や経験により判断していた構造的欠陥の検出や適切なリファクタリング手法の選択を支援する．

6. お わ り に

リファクタリングを支援する機構を提案した．提案した機構により，作業者は，知識や経験に頼ることなくリファクタリングを行える．

従来のリファクタリング作業では，作業者は構造的欠陥を定量的に把握しておらず，作業の成否は作業者の知識や経験に依存するものであった．このため，プログラムの構造的欠陥の箇所を検出することが難しく，構造的欠陥を解消する適切なリファクタリング手法を選択することが難しい，という問題がある．提案した機構は,「構造的欠陥検出基準」という知識を用いて，構造的欠陥を検出する．また,「リファクタリング手法選択基準」という知識を用いて，リファクタリング手法を選択する．さらに，ソフトウェアメトリクスを作業手順の最初と最後で利用することにより，構造的欠陥の程度を定量的に把握でき，リファクタリングの効果を把握しやすくする．これにより，作業者は，知識や経験に頼ることなくリファクタリングを行える．また，提案した機構の実現と適用例として，ソフトウェアメトリクスとしてC&Kメトリクス，リファクタリング手法としてFowlerのリファクタリング手法を用い，ObjectPascal言語で記述されたプログラムについて，その効果を確認した．具体的には，プログラムの構造的欠陥の程度が大きい順に4つのクラスをリファクタリングし，C&Kメトリクスの尺度について偏差値の値を最大で27.3%下げることができた．

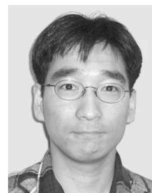
一方，今回の実験では，同一ドメインのプログラムを大量に収集することが難しかったため，ドメインの違う複数のプログラムを用いて構造的欠陥検出基準を作成した．実際の開発現場において，同一ドメインのプログラムを収集し，構造的欠陥検出基準を機械的に生成することは比較的容易であるが，母集団が良質なものであることが必要となる．

残された課題としては，良質な母集団をいかにして大量に集めるか，多くのプログラムへの適用実験，および適用するリファクタリング手法を選択する規則の検討，がある．これらを実際のソフトウェア開発現場と協力して取り組みたい．

参考文献

1) Horowitz and Barry, M.: *Strategic Buying for the Future*, Libey Publishing, Washington DC (1993).
2) Fowler, M.: *Refatoring: Improving The Design of Existing Code*, Addison-Wesley (1999). 児玉公信，友野晶夫，平沢　章，梅沢真史（訳）：リファクタリング：プログラムの体質改善テクニック，ピアソン・エデュケーション (2000).
3) Fenton, N.E. and Pfleeger, S.L.: *Software Metrics*, 2nd Ed., International Thomson Publishing (1997).
4) Chidamber, S.R. and Kemerer, C.F.: A Metrics Suite for Object Oriented Design, *IEEE Trans. Softw. Eng.*, Vol.20, No.6, pp.476–493 (1994).
5) 青木　淳：オブジェクト指向システム分析設計入門，ソフト・リサーチ・センター (1993).
6) Lorenz, M. and Kidds, J.: *Object-Oriented Software Metrics — A Practical Guide*, PTR Prentice Hall Inc. (1994).
7) Abreu, F.B.: Design Quality Metrics for Object-Oriented Software Systems (1995).
8) Opdyke, W.F.: Refactoring Object-Oriented Frameworks, Ph.D.Thesis, University of Illinois at Urbana-Champaign (1992).
9) Harrison, S.R. and Nithi, R.V.: An Investigation into the Applicability and Validity of Object-Oriented Design Metrics, *Empirical Software Engineering*, Vol.3, No.3, pp.255–273 (1998).
10) Beck, K.: *Extreme Programming Explained: Embrace Change*, Addison-Wesley (1999).
11) Opdyke, W.F. and Johnson, R.E.: Refactoring: An aid in designing application frameworks and evolving object-oriented systems, *Proc. SOOPPA '90: Symposium on Object-Oriented Programming Emphasizing Practical Applications* (1990).
12) Roberts, J.D. and Johnson, R.E.: A Refactoring Tool for Smalltalk, *Journal of Theory and Practice of Object Systems* (*TAPOS*), Vol.3, No.4, pp.39–42 (1997).

(平成14年5月14日受付)
(平成15年4月 3 日採録)



秦野　克彦（学生会員）
1975年生．1999年九州大学工学部情報工学科卒業．2001年九州大学大学院システム情報科学研究科情報工学専攻修士課程修了．同年4月より九州大学大学院システム情報科学府博士後期課程に在籍．ソフトウェア工学・ソフトウェア開発環境に興味を持つ．

乃村 能成（正会員）

1969 年生．1993 年九州大学工学部電子工学科卒業．1995 年九州大学大学院情報工学専攻修士課程修了．同年九州大学工学部助手，1996 年九州大学大学院システム情報科学研究科助手．ソフトウェア開発環境・グループウェアに興味を持つ．1994 年情報処理学会奨励賞受賞．

谷口 秀夫（正会員）

1978 年九州大学工学部電子工学科卒業．1980 年九州大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．1987 年同所主任研究員．1988 年 NTT データ通信（株）開発本部移籍．1992 年同本部主幹技師．1993 年九州大学工学部助教授．2003 年岡山大学工学部教授．オペレーティングシステム，実時間処理，分散処理に興味を持つ．著書「オペレーティングシステム」（昭晃堂）等．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員．博士（工学）．

牛島 和夫（正会員）

1937 年生．1961 年東京大学工学部応用物理学科（数理工学コース）卒業．1963 年東京大学大学院数物系研究科修士課程修了．1977 年九州大学教授（情報工学科計算機ソフトウェア講座担当）．1996 年九州大学大学院システム情報科学研究科教授・同研究科長併任．2001 年 4 月から財団法人九州システム技術研究所長．2002 年 4 月から九州産業大学情報科学部教授・学部長併任．本会理事，九州支部長，監事を歴任．現在アクレディテーション委員会委員長．電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE 各会員．1999〜2000 年度 IEEE Fukuoka Section Chair．工学博士．