ソフトウェア工学　136－13
（2002．3．7）

リファクタリング指標の構築に向けた オブジェクト指向システムの漸進的開発における 進化メトリクスのケーススタディ

吉田 正和[1]　蔵川 圭[1]　中小路 久美代[1,2,3]　神谷 年洋[3]

[1] 奈良先端科学技術大学院大学 情報科学研究科

[2] SRA 先端技術研究所

[3] 科学技術振興事業団 さきがけ研究 21

{masaka-y,kurakawa,kumiyo,kamiya}@is.aist-nara.ac.jp

概要

オブジェクト指向ソフトウェアの発展を理解する上でリファクタリングを認識することが重要であると考えられる．ところがソースコードの履歴から自動的にリファクタリングを識別する方法はなく，現状では人間がソースコードを読んで調べる必要がある．本研究ではソースコードの履歴からソフトウェアのバージョン間のソフトウェアメトリクスの変化を調べることによってリファクタリングを識別する手法の構築に向けたケーススタディを行った．ケーススタディではリファクタリングの徴候をとらえていると考えられるいくつかのメトリクスの変化パターンを同定することができた．

A Case Study on Object-Oriented Metrics for Identifying Refactoring Processes in An Evolutionally Development Project

Masakazu Yoshida[1], Kei Kurakawa[1], Kumiyo Nakakoji[1,2,3], Toshihiro Kamiya[3]

[1] Graduate School of Information Science, Nara Institute of Science and Technology

[2] SRA Key Technology Laboratory Inc.

[3] Japan Science and Technology Corp.

{masaka-y,kurakawa,kumiyo,kamiya}@is.aist-nara.ac.jp

Abstract

Identifying refactorings is important to understand evolution of object-oriented software. Since refatctorings would be found only by hand, we conduct a case study on object-oriented metrics toward constructing a methodology to find refactorings by looking at the changes in metrics of available versions. In the case study, we figure out some interesting patterns of changes in metrics which capture the certain aspects of refactoring processes.

1 はじめに

現在，ソフトウェアはますます大規模化する一方で，環境の変化や利用者の要求に応じてその機能を変更していく，いわゆる適応的あるいは発展的な開発プロセスが求められている．

適応的に高品質のソフトウェアを開発する手法として，オブジェクト指向ソフトウェア開発を対象とした，スパイラル型開発 [1] や eXtreme Programming（XP）[2] といった漸進的なソフトウェア開発プロセスが注目されている．このような漸進的ソフトウェア開発プロセスにおいてはリファクタリング [3] — 従来のウォータフォールモデルでは非本質的な後戻りの作業であるとみなされる小規模なソフトウェアの内部設計の変更 — は，本質的な，繰り返し発生しする作業である．従って，このような開発プロセスにおいてソフトウェアの進化あるいは発展プロセスを理解するにはこれらのリファクタリングを認識することが重要であると考えられる．

しかしながら，ソフトウェアの進化あるいは発展プロセスを観察するためにソースコードを人手で調べる [4] には膨大な労力が必要となるため，何らかの自動化が必要である．

そこで，本研究では，オブジェクト指向の漸進的ソフトウェア開発プロセスを対象とした，オブジェクト指向ソフトウェアメトリクスの時系列の変化からリファクタリングを識別する手法の構築に向けて，ソフトウェアの内部設計の変化プロセスをメトリクスの時系列の変化から識別することを目的としたケーススタディを行った．

本ケーススタディにおいては，大規模であること，ソースコードのバージョン履歴が管理されていること，漸進的なソフトウェア開発プロセスによって開発されていること，オープンソースであること，などの理由から，Smalltalk/VisualWorks[5] で記述されたオブジェクト指向グラフィックスライブラリである Jun[6] を対象として観察実験を行った．

2 ソフトウェアメトリクス

本ケーススタディでは，オブジェクト指向ソフトウェアメトリクスとして，基本メトリクス，CK メトリクス，青木メトリクスの 3 種のメトリクスを用いた．後で説明するように，これらのメトリクスが計測するソフトウェアの属性は異なっているため，併用することでより詳しくソフトウェアの変化を識別することが可能になる．

$Vc(c)$	クラス c に定義されているクラス変数の集合
$Vi(c)$	クラス c に定義されているインスタンス変数の集合
$Mc(c)$	クラス c に定義されているクラスメソッドの集合
$Mi(c)$	クラス c に定義されているインスタンスメソッドの集合
$ref_{mi}(m,i)$	メソッド m が変数 i を参照するとき真，それ以外は偽
$ref_{cc}(c,d)$	クラス c がクラス d を参照するとき真，それ以外は偽
$use(c,d)$	クラス c がクラス d のメソッドを使用しているとき真，それ以外は偽
$Loc_{meta}(c)$	クラス c のメタクラス定義のソースコード行数
$loc(m)$	メソッド m のソースコード行数
$size(m)$	メソッド m のバイトコードの大きさ（バイト数）
$calls(m)$	メソッド m に含まれるメソッド呼び出し演算の回数
C_{all}	クラス全体の集合（システムクラスは除く）
$Sup(c)$	クラス c の継承関係における上位クラスの集合
$Sub(c)$	そのクラスの継承関係における下位クラスの集合
$child(c,d)$	クラス d がクラス c の継承関係における直接の子であるとき真，それ以外は偽
$Msg(c)$	クラス c で定義されているメソッドのメッセージセレクタ（シグネチャ）の集合
$Odr(c)$	クラス c のトポロジカルソート順位：C_{all} をクラス間の参照による依存関係によって半順序化したときの順位；最も多くのクラスから部品として参照されているクラスを 1，どのクラスからも部品として参照されていないクラスを $\|C_{all}\|$ とする；定義より Odr は $\|C_{all}\|$ に依存する
$C_{others}(c) = \|C_{all} - \{c\}\|$	
$Msg_{others}(c) = \cup_{d \in C_{others}(c)} Msg(d)$	

表 1: 記号の定義

2.1 定義

表 1 にメトリクスの定義に使用する記号の定義を示す．

基本メトリクス群は Smalltalk/VisualWorks のクラスの基本的な規模を表す 7 個のメトリクスで，定義は表 2 のとおりである．

CK メトリクス群は Chidamber と Kemerer [7] による 6 個のオブジェクト指向複雑度メトリクスであ

LOC(c)	$Loc_{meta}(c) + \sum_{mc \in Mc(c)} loc(mc) + \sum_{mi \in Mi(c)} loc(mi)$
NOA(c)	$\|Vc(c)\| + \|Vi(c)\|$
NCV(c)	$\|Vc(c)\|$
NIV(c)	$\|Vi(c)\|$
NOM(c)	$\|Mc(c)\| + \|Mi(c)\|$
NCM(c)	$\|Mc(c)\|$
NIM(c)	$\|Mi(c)\|$

表 2: 基本メトリクス群の定義

WMC(c)	$\sum_{mc \in Mc} size(mc)/128 + \sum_{mi \in Mi} size(mi)/128$ 注：定数 128 は VisualWorks 2.5 のメソッドのバイドコードサイズの平均値
DIT(c)	$\|Sup(c)\|$
NOC(c)	$\|\{d \in C_{all} \| d \in C_{others}(c) \wedge child(c,d)\}\|$
CBO(c)	$\|\{x \in C_{all} \| x \in C_{others}(c) \wedge (use(c,x) \vee use(x,c))\}\|$
RFC(c)	$\|Mc(c)\| + \|Mi(c)\| + \sum_{mc \in Mc(c)} calls(mc) + \sum_{mi \in Mi(c)} calls(mi)$
LCOM(c)	$LCOM(c) = T_c(c) + T_i(c)$，ただし $Pc(c) = \{(mi,mj) \| mi,mj \in Mc(c), \{i \in Vc(c) \| ref_{mi}(mi,i) \wedge ref_{mi}(mj,i))\}) = \phi\}$ $Qc(c) = \{(mi,mj) \| mi,mj \in Mc(c), \{i \in Vc(c) \| ref_{mi}(mi,i) \wedge ref_{mi}(mj,i))\}) \neq \phi\}$ $Pi(c) = \{(mi,mj) \| mi,mj \in Mi(c), \{i \in Vi(c) \| ref_{mi}(mi,i) \wedge ref_{mi}(mj,i))\}) = \phi\}$ $Qi(c) = \{(mi,mj) \| mi,mj \in Mi(c), \{i \in Vi(c) \| ref_{mi}(mi,i) \wedge ref_{mi}(mj,i))\}) \neq \phi\}$ と定義するとき $T_c(c) = \|Pc(c)\| - \|Qc(c)\|$ $T_i(c) = \|Pi(c)\| - \|Qi(c)\|$ ただし $T_c(c), T_i(c) < 0$ ならば 0 とする

表 3: CK メトリクス群の定義

る．CK メトリクスを Smalltalk/VisualWorks に適用する場合の定義を表 3 に示す．

青木メトリクス群は青木 [8] による Smalltalk で定義されたクラスのクラスライブラリ中の位置を数値化するメトリクスである．青木メトリクス群の定義を表 4 に示す．

HF(c)	クラスの汎化−特化構造指標： $\|Sup(c)\|/(\|Sup(c)\| + 1 + \|Sub(c)\|)$
RF(c)	クラスの全体−部分構造指標： $Odr(c)/\|C_{all}(c)\|$
PF(c)	クラスのポリモルフィズム指標： $\|Msg(c) \cap Msg_{others}(c)\|/\|Msg(c)\|$
CP(c)	クラスの人気度： $\|\{x \| x \in C_{others}(c) \wedge ref_{cc}(x,c)\}\|$

表 4: 青木メトリクス群の定義

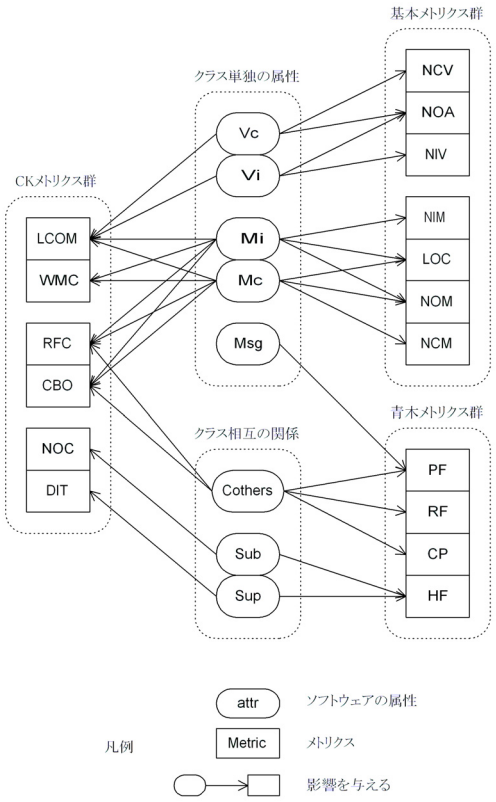


図 1: メトリクスの変化要因

2.2 ソフトウェアの発展とメトリクスの関連性

ソフトウェアのどのような属性の変化がどのメトリクスの変化に関係するかについて整理したものを図 1 に示す．

ソフトウェアの属性については，クラス単独の内部的な変化を表わすグループと他のクラスとの相互的な関係の変化を表わすグループに分類できる．このようにグループ分けをしたことにより，基本メトリクス群のメトリクスはすべてクラス単独の属性のみに基づくこと，青木メトリクス群のメトリクスはすべて何らかのクラス相互の関係に基づいて決定されること，CK メトリクス群のメトリクスには，単独の属性のみに基づくもの，単独の属性と相互の関係の両者に基づくもの，相互の関係のみに基づくもの，の 3 種類があることがわかる．

3 Jun の概要

Jun は，MVC（Model-View-Controller）アーキテクチャに基く純粋なオブジェクト指向で設計され，Smalltalk/VisualWorks で記述された 3D の幾何と

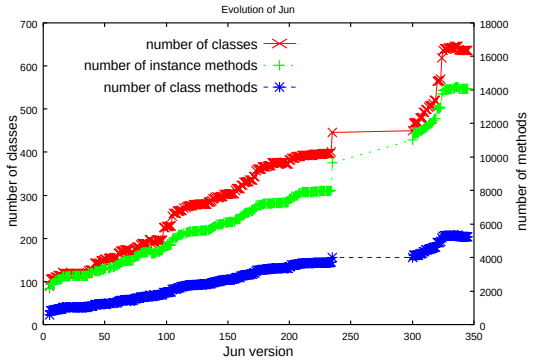


図 2: Jun の発展

位相，マルティメディアデータを扱うためのグラフィックスライブラリである．Jun は，オープンソースのフリーソフトウェアとして GPL（GNU Public Licence）にしたがって配布されている [9].

Jun の各バージョンは，「Jun999」のように，バージョンシリアル番号で識別される．本稿の執筆時点の最新バージョンは Jun415 であり，Jun は過去 5 年間以上にわたりリリースを重ねて成長を続けてきている．このように長期間にわたる Jun のリリース記録はアーカイブに保存されており，Jun がオープンソースのソフトウェアであることから，誰でも自由にソースコードを調べることが可能である．図 2 に，Jun005 から Jun345 までの Jun の発展の概観をクラス数，クラスメソッド数，インスタンスメソッド数の成長グラフとして示す．なお，グラフに不連続な区間があるのは，Jun236 から Jun299 が欠番であるためである．

4　ケーススタディ

今回，Jun005 から Jun234 までの 227 個の各バージョンについて，複数のバージョンにまたがって定義されている 467 個の全てのクラスのそれぞれについて，2 章で説明したメトリクスを適用しその数値の測定をおこなった．各クラスがバージョンアップの時系列に従ってどのように発展してゆくかを分析するため，それぞれのメトリクスの変化を図 3 のようなグラフとしてプロットした．

本章では，これらのグラフをもとに，観察された以下の変化の特徴について考察する．

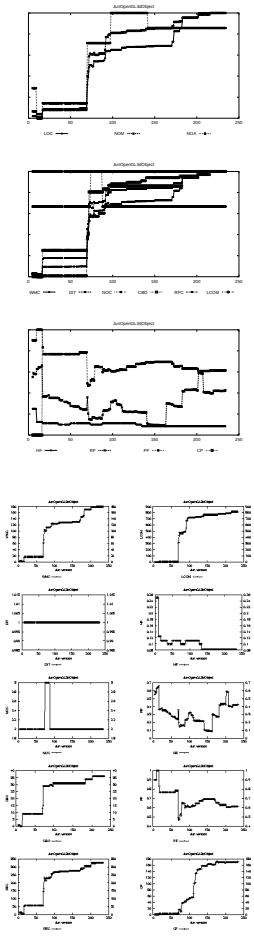


図 3: メトリクス変化グラフの例

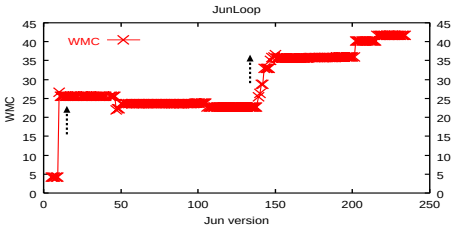
4.1　一般的なクラスの成長に従ってどのようにメトリクスが変化するか

一般的にバージョンアップに伴うクラスの成長は，機能の追加によって生じる．その場合，WMC，すなわちクラスの重み付きメソッド数，RFC，すなわちクラスの反応，および LCOM，すなわちメソッドの凝集の欠如，はいずれも増加する．

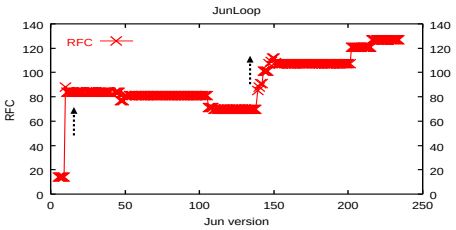
メトリクスの変化パターン

WMC (+)	RFC (+)	LCOM (+)

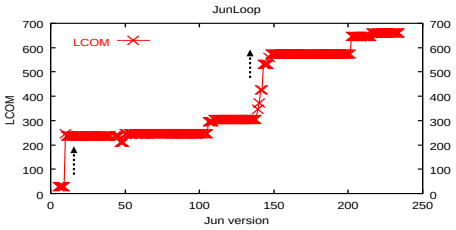
この理由は，機能の追加は一般にそのクラスへのメソッドおよび変数の追加だと考えられるからである．メソッドの追加は，そのクラスの Mc, Mi 集合を増加させるので，これと関連をもつ WMC，RFC が増加する．また，メソッドや変数の変数の追加にともなう Mc, Mi, Vc, Vi 集合の増加により，これと関連をもつ LCOM も変動することが多い．



(a) WMC



(b) RFC



(c) LCOM

図 4: JunLoop のメトリクス変化

このパターンの具体例として JunLoop クラスの WMC，RFC，および LCOM の変化グラフを図 4 に示す．

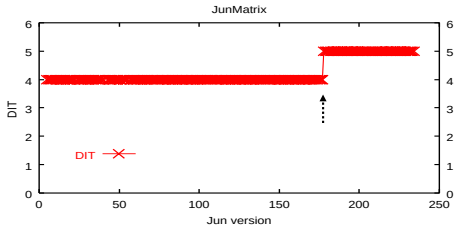
4.2 クラス階層の上位にクラスを挿入した場合の変化

あるクラスの上位に新しいスーパクラスを挿入した場合には，DIT，すなわちクラス階層におけるそのクラスの深さが増加する．これに伴い，WMC，RFC，および LCOM が減少する場合が多く見られる．

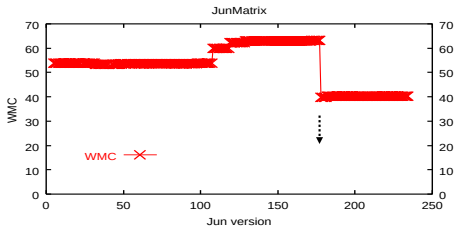
メトリクスの変化パターン

DIT (+)	WMC (−)	RFC (−)	LCOM (−)

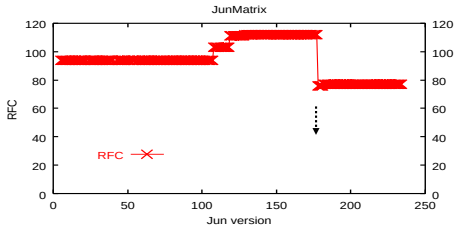
この理由は，クラスの上位に新しいスーパークラスが追加されることにより，そのクラスに定義されていた変数やメソッドの一部が新しい上位クラスに移動するためである．メソッドの移動はそのクラスの Mc, Mi 集合を減少させるので，これと関連をもつ WMC，RFC が減少する．また，変数やメソッドの移動にともなう Vc, Vi, Mc, Mi 集合の減少により，



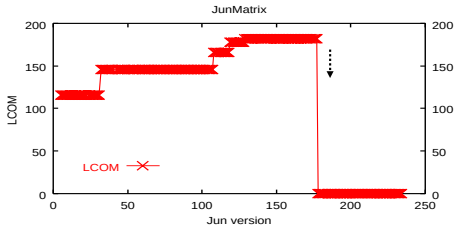
(a) DIT



(b) WMC



(c) RFC



(d) LCOM

図 5: JunMatrix のメトリクス変化

これと関連をもつ LCOM が変動することも多い．

このパターンの具体例として JunMatrix クラスの WMC，RFC，および LCOM の変化グラフを図 5 に示す．この例では，グラフ上の矢印で示す Jun177 から Jun178 へのバージョンアップにおけるメトリクスの変化がこのパターンによるものである．

4.3 上位クラスを削除した場合の変化

あるクラスの上位クラスを削除した場合には，DIT，すなわちクラス階層におけるそのクラスの深さが減少する．これに伴い，WMC，RFC，および LCOM が増加する場合が多く見られる．

メトリクスの変化パターン

DIT (−)	WMC (+)	RFC (+)	LCOM (+)

この理由は，上位クラスが削除されることにより，削除された上位クラスに定義されていた変数やメソッドの一部がそのクラスに移動するためである．メソッドの移動はそのクラスの Mc,Mi 集合を増加させるので，これと関連をもつ WMC，RFC が増加する．また，変数やメソッドの移動にともなう Vc,Vi,Mc,Mi 集合の増加により，これと関連をもつ LCOM が変動することも多い．

このパターンの具体例として JunVrmlCubeNode クラスの WMC，RFC，および LCOM の変化グラフを図 6 に示す．この例では，グラフ上の矢印で示す Jun108 から Jun109 へのバージョンアップにおいて，JunVrmlCubeNode の上位にある 2 つの抽象クラスが削除され，メソッドが JunVrmlCubeNode へ移動されている．

4.4 メソッドの追加による変化

あるクラスにメソッドを追加した場合には，WMC が増加し，これに伴い，RFC，および LCOM が増加する場合が多く見られるが，PF，すなわちクラスのポリモルフィズム指標の変化は増加する場合と逆に減少する場合の 2 つのパターンがあることがわかった．

まず，メソッドを追加した場合に WMC，RFC，および LCOM が増加し，PF が減少するパターンは，ソフトウェアに新しい機能を導入する場合に観察された．

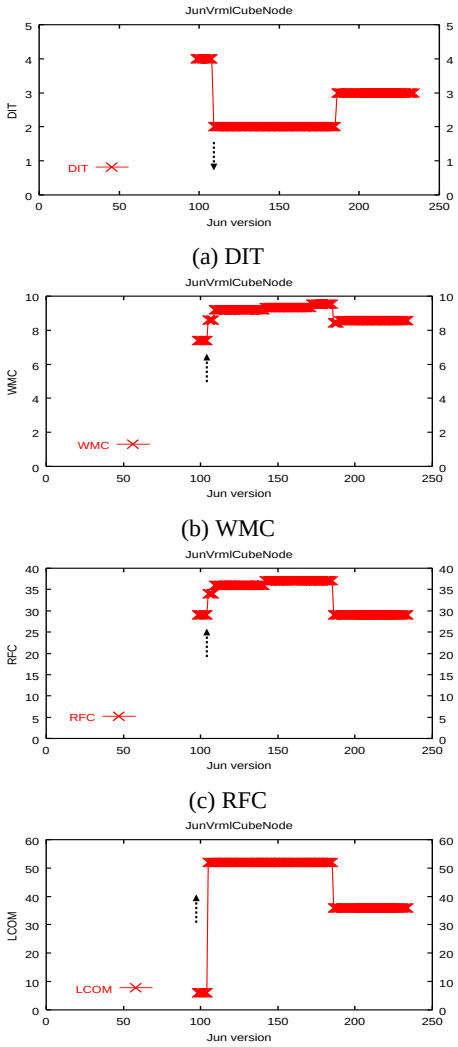
メトリクスの変化パターン

WMC (+)	RFC (+)	LCOM (+)	PF (−)

この理由は，新機能を導入する場合には，クラスに追加されるメソッドのメッセージセレクタ（シグネチャ）は他のクラスで既に定義されているメソッドのどれにも一致しないことが多いからである．このような場合には，Msg，すなわち，そのクラスで定義されているメッセージセレクタの集合は増加するが，$Msg \cap Msg_{others}$，すなわち，他のクラスで定義されているメッセージセレクタとの共通集合は変化しないので，PF が減少する．

図 7 に JunOpenGL3dObject クラスの WMC および PF の変化グラフをこのパターンの具体例として示す．

興味深いことに，図 7(b) の矢印 (A) のように，メ



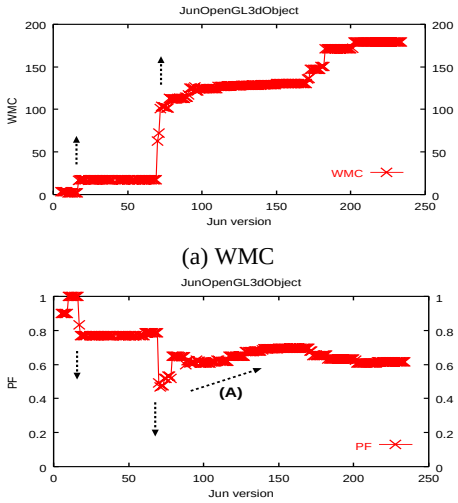
(a) DIT

(b) WMC

(c) RFC

(d) LCOM

図 6: JunVrmlCubeNode のメトリクス変化



(a) WMC

(b) PF

図 7: JunOpenGL3dObject のメトリクス変化

ソッドの追加によって PF が減少した後に，WMC や RFC が一定，すなわち，そのクラスに対するメソッドの変更がないにもかかわらず，PF の漸進的な増加が観察されることが多い．この理由は，もし導入された新機能がソフトウェアの発展に適合的であったときには，その機能は他のクラスにも取り込まれていくからだと考えられる．このような場合には，*Msg*，すなわち，そのクラスで定義されているメッセージセレクタの集合は一定であるが，$Msg \cap Msg_{others}$，すなわち，他のクラスで定義されているメッセージセレクタとの共通集合は大きくなるので，PF が増加する．

次に，メソッドを追加した場合に WMC，RFC，および LCOM が増加し，PF も増加するパターンは，メッセージの共通化のためにクラスにメソッドを追加した場合に観察された．

メトリクスの変化パターン

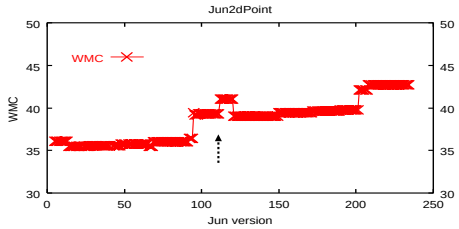
WMC (+)	RFC (+)	LCOM (+)	PF (+)

この理由は，あるクラスにメッセージの共通化のためにメソッドを追加する場合は，そのクラスに追加されるメソッドのメッセージセレクタ（シグネチャ）は他のクラスで既に定義されているメソッドと共有されるからである．このような場合には，*Msg*，すなわち，そのクラスで定義されているメッセージセレクタの集合は増加するが，$Msg \cap Msg_{others}$，すなわち，他のクラスで定義されているメッセージセレクタとの共通集合も増加するので，PF が増加する．

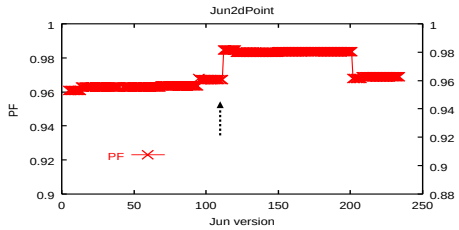
図 8 に Jun2dPoint クラスの WMC と PF の変化グラフをこのパターンの具体例として示す．

4.5 RF の変化パターンによるクラスのグループ化

RF は，*Odr*，すなわちクラス名の参照関係で決定される順序の変化を反映するので，そのクラスに変化がない場合でも，他のクラスからの参照の追加や削除，または，そのクラスが参照している他のクラスの順序に変動があると，それらの影響を受けて大きく変化する．この RF の性質は，メトリクス変化とソフトウェアの変化の関連性を認識することが困難であるので，ソフトウェアの発展プロセスを分析するのには適さないと考えられる．

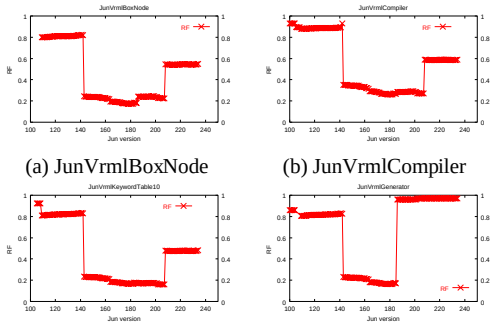


(a) WMC



(b) PF

図 8: Jun2dPoint のメトリクス変化



(a) JunVrmlBoxNode　(b) JunVrmlCompiler

(c) JunVrmlKeywordTable　(d) JunVrmlGenerator

図 9: RF の変化パターン

ところが，多数のクラスについて RF の変化を観察したところ，RF の変化パターンの類似によってクラスをグループ化できる可能性があることがわかった．図 9 に示す例のように，クラス名の参照によって「部品と部品の使用者」の関係にあるクラス (a)(b)(c) では RF の変化パターンに類似性があるのに対し，類似する（同じプリフィックスで始まる）クラス名をもつが関連性が低いクラス (d) では RF は異なる変化パターンをとっている．

この理由は，*Odr* によるクラスの順序付けでは，上に述べた「部品と部品の使用者」の関係にあるクラス群，すなわちクラス名による参照関係があるクラス群は近いところに並び，クラスの順序が変化するときには一緒に移動する傾向があるからだと考えられる．

RF の変化パターンによるクラスのグループ化は，

単独ではリファクタリングにより生じるメトリクスの変化を認識する手法につながるものではないが，単一のクラスより大きな粒度でメトリクスの変化を観察，分析するときの補助的な手段として利用できる可能がある．

5　考察

今回のケーススタディでは膨大なソースコード履歴の中から人手で面白そうなパターンを発見するのには多大な労力が必要であった．また，リファクタリングを識別するためのパターンを網羅できたという確証も得られにくい．開発者の作業イベントやソフトウェアの発展に影響を与えた外部イベントの情報を利用してリファクタリングの候補を絞り込むというアプローチが必要であろう．

Jun には，多くのオープンソースソフトウェアと同様に，開発者メーリングリストが存在するので，このメーリングリストの情報を使って開発者がリファクタリングを行った状況を把握することが可能である．また，Jun は独立したアプリケーションではなく，ライブラリであるので，Jun の発展は Jun を利用する他のシステムの開発の影響を強く受けている [9]．したがって，Jun とこれらのシステム開発との関連も調べていく必要があると考えられる．

6　まとめ

本研究では，ソフトウェアメトリクスの時系列の変化からリファクタリングを識別する手法の構築に向けて，Smalltalk/VisualWorks で記述されたオブジェクト指向グラフィックスライブラリである Jun の 227 個の各バージョンについてメトリクスを計測し，ソフトウェアの内部設計の変化プロセスをメトリクスの時系列の変化から識別することを目的としたケーススタディを行い，観察されたメトリクスの変化の特徴のいくつかについて分析した．しかし，今回論じたメトリクスの変化パターンが起り得るパターンのうちで意味があるもののすべてであるとはいえない．今後は，考察で述べたアプローチも含めたケーススタディを進め，分析を深めることで，最終的には，同定されたパターンを満す変化がリファクタリングを正しく識別することを統計的に検証して，メトリックスの変化パターンをリファクタリングの評価に使えるようにしたいと考えている．

謝辞

本研究のために Jun のソースコードアーカイブを快く提供していただいた SRA 先端技術研究所の青木淳氏に感謝する．

参考文献

[1] Boehm, B.W., A Spiral Model of Software Development and Enhancement, IEEE Computer, Vol.25 No.5, pp.61-72, 1988.

[2] Beck, K., eXtreme Programming eXplained: Embrace Change, Addison-Wesley, 2000.

[3] Fowler, M., Beck, K., Brant, J., and Opdyke, W., Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.

[4] Demeyer, S., Ducasse, S., and Niertrasz, O., Finding Refactorings via Change Metrics, Proceedings OOPSLA’2000, ACM Press, 2000.

[5] http://www.cincom.com/smalltalk/

[6] Aoki, A., Free Software: Jun for Smalltalk – A 3D Graphic Multi-Media Library that supports Topology and Geometry, http://www.sra.co.jp/people/aoki/Jun/

[7] Chidamber, S. R., and Kemerer, C. F., A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, Vol.20 No.6, pp.476-493, 1994.

[8] 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.

[9] Aoki, A., Hyashi, K., Kishida, K., Nakakoji, K., Nishinaka, Y., Reeves, B., Takashima, A., and Yamamoto, Y., A Case Study of the Evolution of Jun: an Object-Oriented Open-Source 3D Multimedia Library, Proceedings of International Conference on Software Engineering (ICSE2001), IEEE Computer Society, pp.524-533, 2001.