論　文

開発履歴中のソースコードを対象とした更新の重要度を評価する手法の提案

横森　励士[†a)]　　野呂　昌満[†]　　井上　克郎[††]

Evaluation Method for Source Code Updates in Software Development History

Reishi YOKOMORI[†a)], Masami NORO[†], and Katsuro INOUE[††]

あらまし　開発対象となるソフトウェアが大規模化するにつれて，進ちょくを把握し管理を適切に行うことが，極めて重要な要素となった．開発途中の生成物から LOC などのメトリックスを抽出し，管理に利用している場面は多く見られる．しかし，開発中のソフトウェアに重大な影響を与える更新を抽出するには，LOC などだけでは不十分で，更新履歴から変更内容を確認し，それによって生じる変化を理解する必要がある．本論文では，ソフトウェア部品間の利用関係の変化の度合を定量化することで，利用関係の変化という観点で更新履歴の中から重大な影響を与えた更新を抽出する手法を提案する．定量化においては，部品間の利用関係に基づいた順位であるコンポーネントランクを用い，更新前後間での各部品の順位の変動を評価基準とする．提案手法に基づきシステムを開発し実際のオープンソースプロジェクトに適用したところ，大規模な機能追加のほかに，コア部品に対する機能追加や，コードの作り直し，リファクタリングなどのメンテナンス作業が抽出できた．これらの更新は，LOC の変化量が少ないものも多く，提案手法はこれらの更新作業の抽出に有益であると考えられる．

キーワード　ソフトウェア部品，コンポーネントランク，更新履歴，メトリックス，Java

1. まえがき

組織におけるソフトウェアプロセスの成熟度モデル CMMI [6] においても見られるように，プロセス改善やプロジェクト管理における計測の重要性は広く認知されているところである．管理や分析においては，構成管理システムなどに蓄積されたプロジェクトデータを分析し，メトリックスに基づいた評価を行う手法が用いられることも多い [7], [10], [12]．このとき，ただ単一のメトリックスからプロジェクトのある側面を計測するだけではなく，複数のメトリックスを用いて多様な観点から計測することが重要となる．例えば，それぞれの更新の重大さを判別するための方法として，一般的に LOC，クラス数などのコードメトリックスを用いて更新の大きさを測ることが多いが，コア部品の改造，リファクタリングやコードの再作成など，大量のコード変更は伴わないが重要な更新も数多く存在する．これらの更新を見つけるためには，更新履歴から変更内容を確認し，それによって生じる変化を理解する必要があり，多くの労力と知識を要する．

本論文では，ソフトウェア部品間の利用関係の変化を定量化することで，利用関係の変化という観点から，更新履歴の中から重大な影響を与えた更新を抽出する手法を提案する [14]．定量化においては，部品間の利用関係に基づいた順位であるコンポーネントランク [5], [11] を用いる．コンポーネントランクは，あるシステム内のソフトウェア部品の直接的，間接的な利用関係を解析することで，各ソフトウェア部品の重要度を評価する手法である．

提案手法では，『利用関係に大きな変化をもたらす更新は，システム全体への影響が大きいという意味で重要な更新である』と考え，『利用関係の変化により，コンポーネントランクも変わる』という仮説を立てている．例えば大規模な機能追加時には，新規に追加される部品が既存のライブラリやデータ構造を利用することで，それらの部品の順位は上がる．これにより，機

† 南山大学数理情報学部，瀬戸市
Faculty of Mathematical Sciences and Information Engineering, Nanzan University, Seto-shi, 489–0863 Japan
†† 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560–8531 Japan
a) E-mail: yokomori@it.nanzan-u.ac.jp

能追加の規模に応じた順位の変動が生じることが予想される．一方でコードの作り直しやリファクタリングの場合，メソッドの抽出や移動，削除，インタフェースの整理などで，既存の利用関係が大きく変化し，それに応じてコンポーネントランクも更新前後で大きく変動すると考えられる．提案手法は，このような利用関係の変化をコンポーネントランクの順位の変動として定量化することで,『利用関係の変化が大きい更新を把握すること』を目的とする．

提案手法に基づき，ある更新におけるコンポーネントランクの順位変動を評価するシステムを実現した上で，実際のオープンソースプロジェクトに適用することで評価実験を行う．評価実験では，提案手法で抽出できたコンポーネントランクの変動が大きい更新が実際にどのような変更内容の更新であるかを確認する．提案手法が,『利用関係の変化が大きい更新を把握すること』に有効であるか，一般的なコードメトリックスを用いた場合に埋もれがちな『コード変更量は少ないが内部的に重要な変化をもたらす更新を抽出する』ために有効な手法であるかを検証し考察する．

以降，**2.** では背景としてコンポーネントランクを紹介する．**3.** では提案手法及び実現したシステムについて述べ，**4.** では適用実験について述べる．**5.** では考察を行い，関連研究を紹介する．最後に **6.** で，まとめと今後の課題について述べる．

2. 部品グラフとコンポーネントランク

一般にソフトウェア部品（Software Comopnent）とは再利用できるように設計された部品とされている [1], [9]．ここではクラスやファイルなどの開発者が再利用を行う単位をソフトウェア部品（あるいは単に部品）と呼び，部品グラフとしてモデル化する．部品グラフ上の頂点はそれぞれの部品を表し，部品間の利用関係を有向辺とする. [5], [11] においてコンポーネントランクモデルを実現した際には，部品の単位を Java のソースファイル（現在はクラス単位）とした．部品間の利用関係については，動的束縛は考慮せずに静的に解析可能な関係を抽出しており，クラスの継承，インタフェース及び抽象クラスの実装，変数宣言及びインスタンスの生成，メソッドの呼出し，フィールド参照を利用関係として部品グラフを作成した．

コンポーネントランクの計算モデルでは，与えられた部品グラフをマルコフ連鎖 [8] でモデル化し，マルコフ連鎖の定常分布を計算する．まず部品グラフ上の各頂点の重みの総和が 1 になるように各頂点の重みの初期値を与える．マルコフ連鎖の場合，計算対象のマルコフ連鎖が既約であれば初期の状態によらず定常分布に収束するので，各頂点に均等な値を割り振っている．その後，以下の計算を収束するまで繰り返すことで，各頂点の重みを計算する．実際には各頂点の重みは，それぞれの頂点から頂点への分配率を表した行列における，固有値 1 の固有ベクトルとなる．

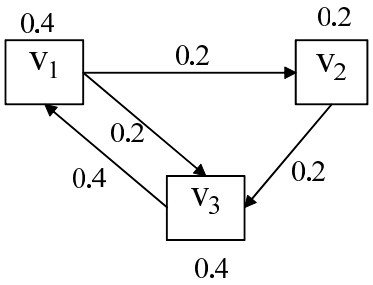


図 1　重みの計算例
Fig. 1　An example of stable weights assigned to nodes and edges.

(1)　各頂点の現在の重みをその頂点を始点とする有向辺に分配する．分配率をもとに重みの分配を変えることはできるが，基本的には等分配する．

(2)　(1) で決定したそれぞれの有向辺の重みをもとに，各有向辺の終点の新たな重みを決定する．

図 1 は，与えられたグラフにおいて各頂点の重みを計算した結果である．v_1 は二つの有向辺の始点で，v_1 の重み 0.4 は二つの辺に等分されている．また，v_3 は二つの有向辺の終点で，それぞれの辺が 0.2 の重みをもつため，v_3 の重みは 0.4 であることが分かる．

この定常分布における，それぞれの部品に対応する各頂点の重みで順位付けを行い，その順位がその部品の集合におけるコンポーネントランクとなる．モデルの実現 [5], [11] においては，部品グラフでモデル化されるマルコフ連鎖を既約な連鎖にするために，すべての頂点間を非常に小さな配分率の擬似辺で連結している．また，コピーされた部品を考慮するために，コピーされたと判断できるような部品は一つの部品群となるように部品群化を行っている．実験からは，よく利用されるクラスや一般的な使い方をしているクラスが集中して上位に現れるなど，コンポーネントランクは利用実績を十分に表す指標であることが示されている．また，一部の頂点が非常に高い重みをもち，ほとんどの部品の重みは低くなる傾向が知られている．

このコンポーネントランクは，ソフトウェア部品検索システム SPARS-J [4] において検索結果の順位付けに用いられている．コンポーネントランクを用いることで検索結果全体がユーザの想定する順位付けに近くなるなど，ソフトウェア部品検索に有用なものであることが評価実験で示されている．また，関連研究として非常に高い重みをもつ部品に着目することでソフトウェアの理解を容易にするための研究 [2] や，クラスタリングに頂点の重みの要素を加味することで精度を向上させる研究 [3] などが行われている．

3. コンポーネントランクの変化に基づく評価手法

ソフトウェアが完成するまでの過程では，機能，データや制御構造を実現する部品が未完成なソフトウェアに徐々に組み込まれていき，完成に近づいていく．利用関係の観点から見た場合，部品の追加に合わせて利用関係も変化していく一方で，完成に近づくにつれて利用関係が固定化されていくとも考えられる．これらの部品や利用関係の変化は，ソースコードを読み進めていくことで把握可能であるが，この作業は大きな手間と知識が必要なため，把握できる人間は限られる．

このような変化の大きさなどの情報を定量化することは情報の共有の観点からも有益であると考えられる．一般的に，LOC，クラス数などのコードメトリックスを用いて更新の大きさを測ることが多いが，これらのメトリックスは常に単調増加であることが多く，機能追加などの変更量の多い更新しか把握できない．実際には，コア部品の改造，リファクタリングやコードの再作成など，大量のコード変更は伴わないが重要な更新も多く，システム全体に影響を与えるような重要な更新を見逃してしまう可能性がある．

本論文では，開発プロセス中の利用関係の変化に着目し，コンポーネントランクを用いて利用関係の変化度を定量化する手法を提案する．提案手法では，開発プロセス中の更新があった各時点でのソースコードを取得する．それぞれの更新に対して，更新前，更新後のコンポーネントランクをそれぞれ求める．この二つのランクに共通して存在する部品についてそれぞれの順位の差を求め，その平均をもとに更新の影響度を調査する．

提案手法においては『利用関係に大きな変化をもたらす更新は，システム全体への影響が大きいという意味で重要な更新である』と考え,『利用関係の変化に応じて，コンポーネントランクも同様に変化する』という仮説を立てている．例として，ソフトウェアに新たな機能が追加される場合を考える．機能の追加時には，新規に追加される部品が既存のライブラリやデータ構造を利用することが予想される．そのとき，それらの部品の順位は上がると予想できる．機能追加が大規模である場合は，順位が上がる部品が増え，順位の変動が大きくなると考えられる．

二つ目の例として，コードの作り直しやリファクタリングの場合を考える．このときメソッドの抽出や移動，削除，インタフェースの整理などで，既存の利用関係が大きく変化することが予測される．部品間に新しい利用関係が生じる場合と同時に，利用関係が削除される場合も想定できる．そのため順位が上がる部品，下がる部品が同時に発生し，大きな変動が起こると考えられる．更にこれらの変更がコア部品に近ければ近いほど，上位の部品がもつ大量の重みが他の部品に流れることで順位の変動が大きくなると考えられる．

このように提案手法は大規模な機能追加だけでなく，利用関係の変化が大きい更新や，システムの中心に近い箇所への変更を含む更新にも高い値を示すと予想される．提案手法は，開発履歴などを分析する際に大量の更新の中から，ソフトウェアに大きな影響を与える更新を抽出するために適した手法であると考えられる．

3.1 計算プロセス

提案手法では，ある更新 U の影響の大きさ $impCR(U)$ を，更新前，更新後の二つのリビジョン間のコンポーネントランクにおける個々の部品の順位の変動を用いて定量化する．更新前の部品の集合を $S_{pre}(U)$，更新後の部品の集合を $S_{aft}(U)$ と定義する．更に，二つのリビジョン間に共通して存在する部品を $S_{uni}(U)$ と定義する．

$$S_{uni}(U) = \{X | X \in S_{pre}(U) \wedge X \in S_{aft}(U)\}.$$

影響の大きさ $impCR(U)$ を以下の手順で計算する．

(1) $S_{pre}(U)$，$S_{aft}(U)$ に対しコンポーネントランクを計算する．それぞれの部品の数が異なる可能性があるため，得られた順位は正規化する．

[定義 1]（部品 X の順位の正規化）

$S_{pre}(U)$ に属する部品の数を $N_{pre}(U)$，$S_{pre}(U)$ に属する部品 X の更新前のコンポーネントランクにおける順位を $CR_{pre}(U, X)$ とする．このとき，次の式で部品 X の順位を $NCR_{pre}(U, X)$ として正規化する．正規化により 0 から 1 の値となり，1 が最上位を表す．

更新後の順位についても同様に正規化し，それぞれの部品 X ごとに $NCR_{after}(U,X)$ を求める．

$$NCR_{pre}(U,X) = \frac{N_{pre}(U) - (CR_{pre}(U,X) - 1)}{N_{pre}(U)}$$

(2) $S_{uni}(U)$ に属する部品それぞれについて，更新前後の正規化された順位の差を求め，その絶対値の平均をある更新 U の影響の大きさとする．

［定義 2］（更新 U の影響度の大きさ $impCR(U)$）

$S_{uni}(U)$ に属する部品の数を $N_{uni}(U)$ とし，$S_{uni}(U)$ に属する部品 X の正規化後のコンポーネントランクを $NCR_{pre}(U,X)$（更新前），$NCR_{aft}(U,X)$（更新後）とする．このとき，U の影響度 $impCR(U)$ を次の式で定義する．

$$impCR(U) = \frac{\sum_{X \in S_{uni}(U)} |NCR_{aft}(U,X) - NCR_{pre}(U,X)|}{N_{uni}(U)}.$$

3.2 評価システムについて

提案手法をもとに，開発履歴における各更新の影響度を計算するシステムを構築した．本システムは，Java で記述され，CVS でソースコード管理が行われているプロジェクトを対象とする．本システムの構成図を図 2 に示す．本システムでは，部品解析時に SPARS-J の部品登録部を利用する．そのため，部品及び利用関係の定義は SPARS-J における定義 [4] と同一である．解析の手順は以下のとおりである．

(1) 更新日リストの作成：CVS のヒストリを利用して，更新日時の一覧を取得する．現在は更新の粒度を日単位としており，同一日に複数の更新があった場合は，まとめて一つの更新とみなす．

(2) コンポーネントランクの計算：更新日ごとにチェックアウトコマンドを用いて，ソースコードを取得する．取得したソースコードを更新日別に SPARS-J を用いて解析し，各時点でのコンポーネントランクを計算する．

(3) 影響度メトリックスの計算：各更新日におけるコンポーネントランクから，各更新の影響度を計算する．

(3-1) それぞれの更新日 U ごとに，更新前，更新後のコンポーネントランクを取り出す．

(3-2) **3.1** の定義に基づき，$impCR(U)$ を計算する．

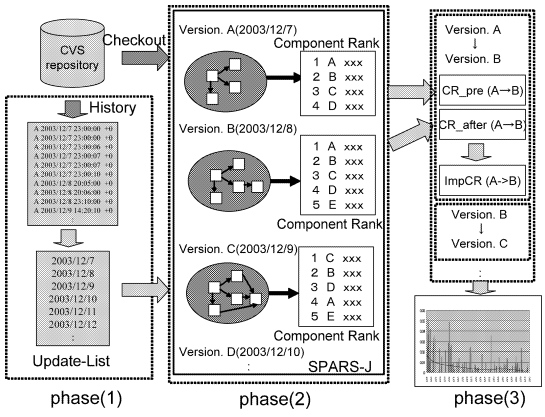


図 2 評価システムの構成
Fig. 2 Architecture of the evaluation system.

4. 評価実験

本章では，提案手法をオープンソース開発プロジェクトに適用した結果を紹介する．個別のプロジェクトに対する適用結果とともに，全体的な傾向として複数のプロジェクトにおける適用結果をまとめ，分類した結果を紹介する．

4.1 実験の目的

提案手法は,『利用関係に大きな変化をもたらす更新はシステム全体に影響を与えるような重要な更新で，そのような更新はコンポーネントランクを大きく変動させる』という仮説のもとに成り立っている．評価実験では，提案手法で抽出できたコンポーネントランクの変動の大きい更新が実際にどのような変更内容の更新であるかを確認する．提案手法が,『利用関係の変化が大きい更新を把握すること』に有効であるか，一般的なコードメトリックスを用いた場合に埋もれがちな『コード変更量は少ないが内部的に重要な変化をもたらす更新を抽出する』ために有効な手法であるかを検証する．

4.2 実験内容

解析対象として，SourceForge [16] に登録されているプロジェクトを利用した．SourceForge はオープンソース開発者向けのコミュニティで，大量のソフトウェア開発プロジェクトが登録されている．大量の開発プロジェクトの中から，(1) 開発言語が Java である，(2) ソースコード管理に CVS を用いている，(3) 開発履歴やコメントが十分蓄えられている，という条件で表 1 の 12 プロジェクトを選び，適用対象とした．

プロジェクトごとに，提案手法を適用し，得られた

結果をグラフ化した．それぞれのプロジェクトは開発規模や開発期間がまちまちであるため，システムが要した解析時間も多種多様である．開発期間が比較的短いプロジェクトは約 20 分（1 リビジョン当り約 20 秒）ほどであったのに対して，開発期間が長く大規模なプロジェクトには，約 10 日（1 リビジョン当り約 10 分）の日時を必要とした．以下では，あるプロジェクトに対する適用結果を紹介した上で，適用結果における全体的な傾向を述べる．

4.3 適用の結果

ここでは，Jbidwatcher [20] に対する適用結果を紹介する．Jbidwatcher は，オークションの監視を支援するためのツールである．2000 年 5 月から 2006 年 1 月までの開発期間において，303 回（日）分の更新があり，解析対象とした．当該区間における一般的なメトリックスの推移を図 3 に示す．それぞれのメトリックスの値が単調に増加していることから開発がコンスタントに続いていることが分かるとともに，これらのメトリックスは非常によく似た増加傾向を示していることが分かる．

図 4 は *impCR*（影響度）とクラス数の推移を示したものである．グラフ中では，クラス数の推移を灰色の線，*impCR* についての多項式近似曲線を点線で示している．クラス数は図 3 のメトリックスと同じように推移していることが分かる．一方で，*impCR* の値は，減少しながら推移していることが確認できる．

図 5 は開発の前期，中期，後期における *impCR* の分布をメトリックスグラフで示したものである．一番左側の軸から，開発時期，コンポーネント数，LOC，コード増減量，*impCR*（影響度）の値を表している．開発が進行するにつれて一番右の値の分布幅が狭くなっており，利用関係の変化が収束している様子をうかがうことができる．

表 2 は高い *impCR*（影響度）の値を示した更新の一覧である．その変更内容については，CVSweb などの情報をもとに筆者が調査した．開発の初期における影響度の高い更新として，ブラウジング機能やダンプ機能の追加，パスワードログインへのサポートなどのような，システムを横断する機能や，基本的な機能やデータ構造の追加などを検出できた．

その一方で開発の中期以降に関しては，コードの再作成，インタフェースの整理，リファクタリングなどのクラスやデータの構造を変える変更を含む更新を，影響度の高い更新として検出できた．表 2 のほかに

表 1　適用対象プロジェクト
Table 1　Target projects.

	Name	From	To	Revision	LOC
1	azureus	2003/7	2006/2	903	374 K
2	freemind	2000/8	2006/2	309	23 K
3	galleon	2005/2	2006/2	55	91 K
4	ganttproject	2003/5	2006/2	543	73 K
5	jasperreports	2003/12	2006/2	349	149 K
6	jbidwatcher	2000/5	2006/1	303	29 K
7	jedit	2000/1	2006/2	1515	830 K
8	megamek	2002/2	2006/2	990	107 K
9	openwfe	2003/6	2006/2	643	81 K
10	pmd	2002/6	2006/2	802	124 K
11	pydev	2003/7	2006/2	315	45 K
12	xui	2003/3	2006/2	68	153 K

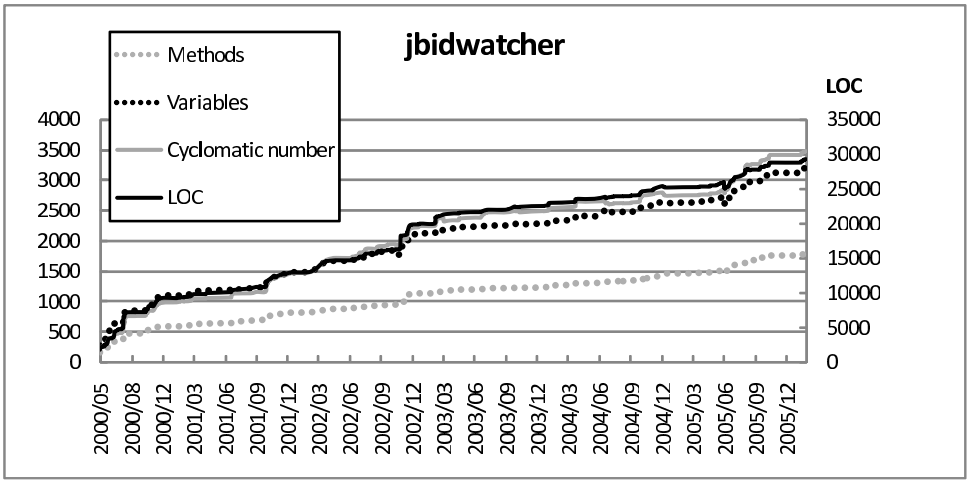


図 3　一般的なメトリックスの推移
Fig. 3　Transition of general metrics.

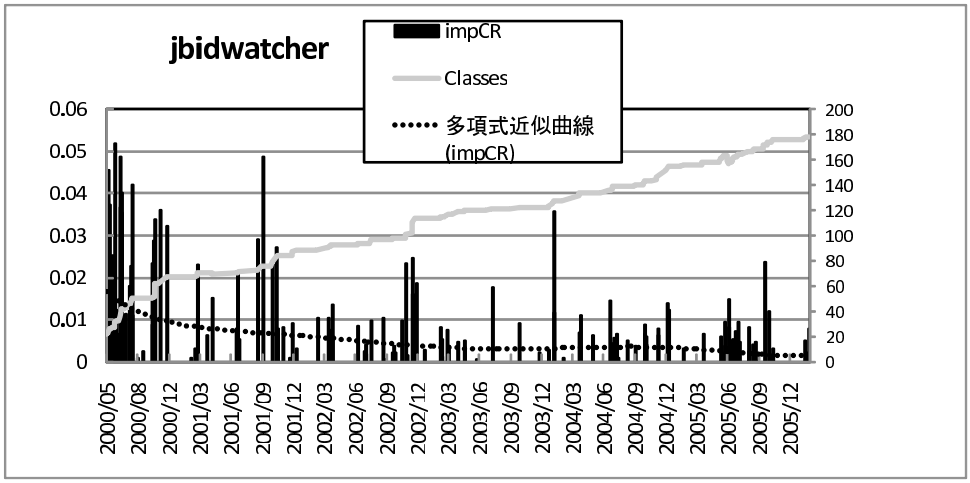
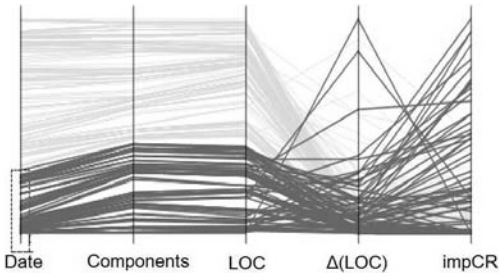
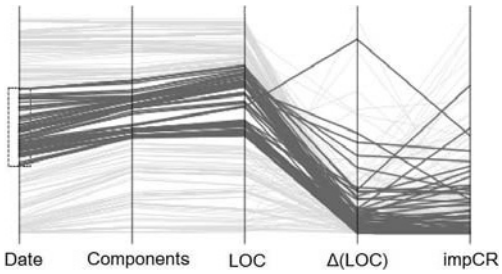


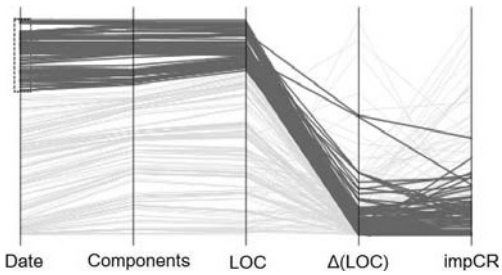
図 4　部品数と *impCR* の推移
Fig. 4　Transition of *impCR* and number of components.



(a) 開 発 初 期



(b) 開 発 中 期



(c) 開 発 後 期

図 5　開発時期別に見た *impCR* の分布
Fig. 5　Distribution of *impCR*.

表 2　*impCR* の高い更新一覧
Table 2　High *impCR* updates.

	Date	*impCR*	Major Contents * Implementation
1	2000/6/23 (15th)	0.0516	Adjust mouse action Delete meaningless codes
2	2001/8/31 (70th)	0.0488	Restructure of UI Remove generic constants
3	2000/7/11 (20th)	0.0486	Handle a proxy server Add generic constants
4	2000/6/3 (3rd)	0.0454	* Browsing
5	2000/8/12 (34th)	0.0420	Extract logging classes * Visual display
6	2000/7/13 (22nd)	0.0400	Password login support * Bidding features
7	2000/6/6 (5th)	0.0372	* Splash screen
8	2000/7/8 (19th)	0.0367	* Menu bar
9	2000/11/4 (48th)	0.0359	Use Auction Manager * Multiple tabs
10	2004/1/4 (164th)	0.0352	Clean up platform -specific code

も，一般的なインタフェースの整理，テーブルや検索機能のリファクタリングなどを含む更新を検出できた．

表 3 は，提案手法が最も変動が大きいと判断した 2000 年 6 月 23 日の更新において，順位の変動が激しい部品をリストアップしたものである．この日の更新では，主にマウスアクションの調整が行われており，JBidMouse や，JMouseAdapter などのクラスの順位が大きく上がっている．また，開発ログにはオークションクラスの追加・更新・削除を扱うクラスが JBidWatch から JBidMouse に移行したことが示されており，JBidMouse が内部的に重要な部品に移

表 3　2000/6/23 の更新における主な部品の順位変動
Table 3　Key components in update of 2000/6/23.

	Class	before	after	diff
1	JBidMouse	0.16	0.48	+0.32
2	JMouseAdapter	0.38	0.64	+0.26
3	TableSorter	0.16	0.38	+0.22
4	AuctionTableModel	0.16	0.28	+0.12
5	AuctionInfo	0.77	0.8	+0.03
:	:			
50	JBidWatch	0.44	0.41	−0.03
51	JDropHandler	0.47	0.44	−0.03
52	ebayServer$ebayAuction	0.25	0.22	−0.03
53	JHTML$htmlToken	0.81	0.77	−0.04
54	JDropListener	0.38	0.25	−0.13

行したことが分かった．このように，コンポーネントランクは利用関係上重要な部品を上位に順位付けでき，更新によって重要な部品が発生した場合は，更新後のその部品の順位もそれに合わせて上昇していることが分かる．

一方，全修正の約半分（159 件）の更新において，*impCR* の値は 0 であった．そのような *impCR* の値が低い更新の内容を調べてみると，それらの更新は次のような更新であった．

(1)　ソースコードにはまったく影響のない修正

そのような更新の例として，ドキュメントや TODO の追加，データファイルの追加，タブの削除のような整形やコメント追加などの修正などが確認できた．

(2)　他のクラスに全く影響を与えない変更

そのような更新の例として，メソッド内部のバグの修正，単体テスト用のデバッグコードの追加，クラス内部での処理方法の変更，クラス内で完全に閉じたリファクタリングなどが挙げられる．

(3)　単独クラスの変更で完全に吸収できる機能追加

既に実現された機能を拡張することで，対応可能な種類を増やした場合などが当てはまる．例として，対応可能な設定ファイルの追加や，システム外の要素との入出力機能の追加などが挙げられる．

(4)　既に利用関係が存在するクラス間での機能追加

参照するフィールドの追加や，既存の機能と同様な処理方法によって実現可能な機能の追加のように，既に利用関係が存在しているクラス間の上で別の機能を実現した場合を指す．各部品の間に利用関係が存在するかどうかでグラフ上での辺の有無を決定しているため，更新前後で部品グラフの形は変化しない．

これらのことから，既存の部品によって作られている枠組みの中で，枠組みを壊さずに修正をする範囲では，*impCR* の値は 0 若しくは非常に低い値となることが分かる．これらの変更は局所的なものがほとんどで，システム全体の利用関係に影響を与えるものではなく，利用関係の変化という観点から見た場合さほど重要なものではないと思われる．

4.4　全体的な傾向

すべてのプロジェクトに対して当てはまることとして，jbidwatcher の結果に見られるように，開発が進行するにつれて *impCR*（以下，影響度）の値がだんだん減少していった．開発初期の段階では，必要な機能の実装により利用関係の変化が起こりやすく，高い影響度の値を示すが，ソフトウェアの機能が充実するにつれて影響度の値は徐々に下がっていく．このことから，開発の進行により必要な機能やデータが固定化されるのと同様に，利用関係も固定化されていくことが推測できる．

また，オープンソースプロジェクトらしい特徴として，いくつかのプロジェクトにおいて，影響度についての多項式近似曲線が，減少しながらも波を打った形状になった．このことは，短いサイクルで機能追加が活発に行われることが原因であると思われる．

一方で，各更新において最も順位が変動した部品の変動幅も確認したが，こちらは開発の時期にあまり影響されない値であった．このことは，更新の間隔を日としたことで 1 回 1 回の更新の規模は開発の時期を通じてあまり変わらないことを示していると考えられる．一般に開発中は常にプログラムサイズが単調増加するため，更新の規模に対してプログラムサイズが大きくなっていく．このことも，開発が進行するにつれて影響度の値がだんだん減少していく要因の一つであると考えられる．

影響度 (*impCR*) の値は前述のとおり開発の進行により減少していくため，前後の更新と比べて相対的に高い値を示している更新を抽出した．その変更内容を CVSweb などの情報をもとに調査したところ，抽出された更新には単なるバグ修正はほとんど見られず，そのほとんどは以下に示す 4 種類の変更内容に属する更新であった．

(1)　大規模な機能追加

この種の更新では，既存のライブラリが利用され順位が上昇すると同時に，新たにコア部品が追加される．その一方で，既存の部品間の利用関係はあまり変化しない．そのため，開発の初期では，ライブラリ関係の部品順位の上昇やコア部品の追加による変動の影響が非常に大きく現れ，高い影響度を示す更新となる．し

かし時間の経過により既存の部品間の関係が固定化されるため，この種の更新はさほど高い影響度を示す更新ではなくなる．

（例） galleon [17] は TiVo HME プロトコルを利用したホームメディアサーバである．開発の過程において，天気情報，音楽プレイヤー，オンラインラジオ，フォトビューア，iTunes への対応など様々な機能追加がなされている．開発の初期はこれらの更新は高い影響度を示す更新であったが，それぞれの機能は比較的独立なものであるため，開発が進行するにつれて機能追加による更新の影響度はだんだん低くなっていった．

(2) コア部品への変更を含む機能追加

コア部品への変更を含む機能追加の場合，コア部品への利用関係が増加したり，コア部品のもつ大量の重みの配分先が変わったりするため，順位全体への影響が大きくなる．この種の更新は，開発初期だけでなく中期以降においても高い影響度をもつ更新として確認することができた．

（例） PMD [18] は Java ソースコードの潜在的な問題を指摘するツールである．開発の後期において非常に高い影響度を示した更新として，Java1.5 の文法や JSP への対応を目的とした AST に対する機能追加を含む更新を確認できた．これらの更新では，多数のコア部品を含む構文解析部の修正が確認でき，実際に影響が大きい更新であると考えることができる．

(3) システムを横断する機能の追加

デバッグ機能やログ機能などシステムを横断する機能の場合，新たに追加された部品にシステム全体から利用関係が追加されるため，順位全体への影響が非常に大きい．この種の更新は (2) の場合と同様に，開発初期だけでなく中期以降においても高い影響度をもつ更新として確認することができた．

(4) 作り直しやリファクタリングなどの大規模なコード整理

ここでいうコード整理とは，クラス内部で閉じているような小規模なものでなくシステム全体やサブシステム全体，データ構造に影響が及ぶような大規模なものを指す．この種の更新はメソッドの抽出や移動，削除，インタフェースの整理などを伴うため，既存の利用関係が大きく変動しやすい．そのため順位が上がる部品，下がる部品が同時に大量に発生すると同時に，コア部品の消滅，誕生なども想定され，順位全体への影響も非常に大きい．この種の更新は，開発中期以降において時折見られるが，コア部品を整理対象に含む場合は，特に高い影響度をもつ更新として確認できた．

（例） GanttProject [19] はガントチャート作成用の GUI ツールである．2005 年の開発において相対的に高い影響度を示した更新として，モーションリスナー，マウスイベント，入力ファイル，リソースなどへのリファクタリングを含む更新を確認できた．また，その他の区間では，利用関係やデータ構造へのリファクタリングも高い影響度を示した更新として確認できた．

以上のように，提案手法により利用関係に大きな変更をもたらす更新だけでなく，中心的な機能を果たす部品への更新も影響度が高い更新として見つけ出すことが容易になると考えられる．

5. 考　　察

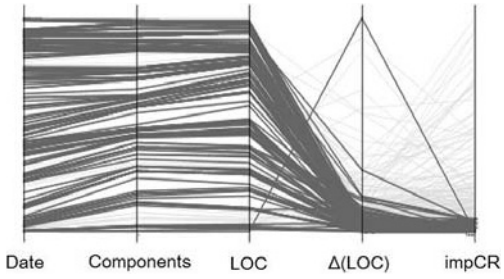
5.1 他のメトリックスとの比較

開発プロセスを分析する際に一般的によく利用される方法は，LOC の推移を見て，過去や現在の進ちょくが想定どおりか，今後目標（納期など）を達成するために何か対策が必要かなどを確認するという方法である．分析の際，他のメトリックスも用いられることもあるが，図 3 を見ても分かるとおり，その他の量を表すコードメトリックスも LOC とほぼ同じ傾向を示すことが多く，それほど大きな意味をなさない．

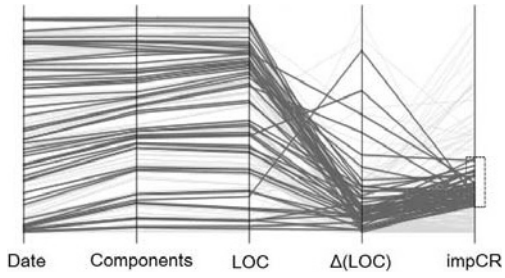
重要な更新を見つけ出す際には，開発者の主観によりピックアップを行う方法や，コードの変更量から規模を推測し，その規模に応じて影響度を見積もる，という方法が手軽で実際の分析ではよく用いられている．本論文が提案しているような，利用関係の変化から影響度の高い更新を見つけ出すということは過去に研究の対象になっていない．

実際，コードの変更量を用いることで，コードの変更量の大きい更新を抽出することができるが，ソフトウェアに大きな影響をもたらす更新は，必ずしもコードの変更量は大きくない．例えばリファクタリングなどのコード整理を含む更新の場合，一般的に機能追加に比べてコードの変更量は小さいが，更新前後で処理の実現方法が変わり得るなど，重要な意味をもつ更新であるといえる．

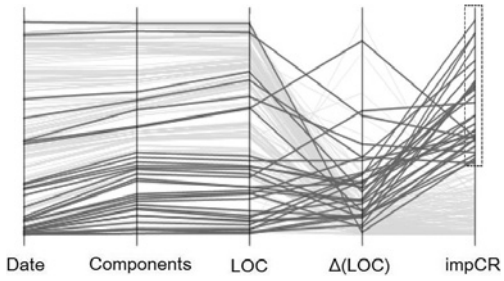
図 6 は，jbidwatcher におけるコード変更量（右から二つ目）と *impCR*（一番右）の関係を表したメトリックスグラフである．*impCR* の小さい更新は，基本的にコード変更量も小さく，どちらの観点でもあまり大きな影響は与えないといえる．その一方で，*impCR*



(a) *impCR* の値が低い更新



(b) *impCR* の値が中程度な更新



(c) *impCR* の値が高い更新

図 6　*impCR* の値（一番右）別に見たコード変更量（右から 2 番目）の分布
Fig. 6　Distribution of an amount of code changes.

が一定以上ある更新は，コード変更量はまちまちで，コード変更量があまり大きくない更新も多い．このような更新は，リファクタリングやコード整理，コア部品などへの機能追加などを含んでいることが多く，分析時に提案手法を用いることでこれらの更新に脚光を当てることができるようになる．

5.2　コンポーネントランクの推移について

提案手法では，更新前後での各部品の順位の変化の平均という形で，各更新の影響度を求めている．コンポーネントランクを利用したもう一つのアプローチとして，各部品の順位の推移をグラフ化するという方法が考えられる．

図 7 は，jbidwatcher において上位 10 位に入ったことがある部品のコンポーネントランクの推移を表したものである．開発の初期は変動が激しいが，中期以降は順位が安定していく様子がうかがえる．図 7 からは中期以降において順位の変動が激しい時点がいくつか見受けられるが，その時点ではすべてコードの作り直しや整理が行われていた．通常このような場面ではその後の拡張性などを考慮して作り直すことが多く，更新後に順位の変動が止まっていることから，これらの更新が成功に終わったことがうかがえる．

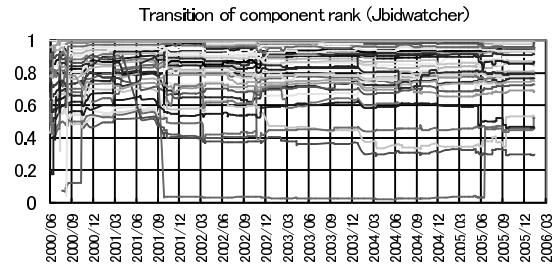


図 7　コンポーネントランクの推移
Fig. 7　Transition of Component Rank.

5.3　提案手法の利用法

提案手法をプロセス管理に用いる際には，いくつかの利用方法が想定できる．一つ目の利用方法は，実験で行ったような事後分析若しくは進ちょくの確認作業において，*impCR* の高い更新を抽出するという方法である．評価実験における結果からは，*impCR* の高い更新が大規模な機能追加や，コア部品への部品追加，コードの作り直しなどの作業であることが多いという結果が得られた．そのような更新の前後では，利用関係やデータの流れが大きく変わりやすいと考えられ，既存システム内の利用関係が大きく変わったかどうかという観点から，開発期間の中で生じた更新の重要性を裏付けする指標として *impCR* を利用可能であると考えている．例えば進捗報告などの場において，利用関係が大きく変わるような更新の存在を浮かび上がらせ，そのような更新が存在することを開発者や管理者の間で把握し，その内容を確認し合うことは，開発における現状の認識を共有する際に非常に役立つと考えられる．

二つ目の利用方法は，更新の *impCR* の値の大きさを予想し，実際に計算された値と比較することで，現状が想定どおりかを確認するというものである．評価実験の結果からは，比較的短いサイクルで繰り返し開発を行うようなオープンソース開発であっても，*impCR* の値は波を打ちながら徐々に収束するという結果であった．ウォーターフォール型の開発モデルの

場合，設計時に必要な機能がある程度固まっているという状態が想定でき，機能の実装が一通り終わった時点でテストフェーズに移行し，*impCR* の値は収束していくと考えられる．このような想定のもとに，例えば開発の後期における更新の *impCR* を計測することで，既存システムの利用関係が大きく変わるような変更が起きているかを把握する．開発が順調な場合，ソフトウェアの出荷間際や納期間近のときにはほぼシステムは完成し，利用関係もほぼ収束している状況が想定できる．このような想定の一方で，開発の末期に *impCR* の高い更新が頻繁に検出できた場合を考える．この場合，利用関係がまだ収束しておらず,．開発対象のシステムがまだ未完成な状態であったり，動いているとしても，検証が十分でなくテストが不足しているなどの状況が想定される．提案手法をこのように用いることで，利用関係の面からリスクの高い状況であるかどうかを確認することができると考えている．

三つ目の利用方法は，*impCR* の収束状況や，*impCR* の値が高い更新などを分析することで，コードの整理の必要性を判断するというものである．システムを横断する機能の追加や，不適切な機能追加の方法などで，システムに機能が追加されていくにつれ，当初の設計構造はだんだん崩れていき，部品の独立性は徐々に低下していく．このようなシチュエーション下で，更に機能を追加しようとすると，あちこちの部品に記述を加えることが必要であるという状況になりやすい．この状況下で機能追加を行うと，既存の部品間に多数の利用関係が新たに引かれやすくなり，*impCR* の値が高くなりやすいと推測できる．開発の中期以降に，単なる機能追加の場合でもその更新の *impCR* の値が高いというような場合が頻発する場合，部品の独立性が低いからという理由も十分に考えられる．実際にコードの再構築やリファクタリングなどを行う際には，コードの独立性を低くしている原因をきちんと追究し，それらの必要性があるかを判断する必要がある．これらの行動を行うためのきっかけとして，*impCR* の値が利用できるのではないかと考えている．

また，部品の独立性を低下させている原因が，多数のシステムを横断する機能であるならば，開発言語を AspectJ のようなアスペクト指向言語に切り換えて，横断している機能をアスペクトとして切り分けるなどの決断をすることで，ソースコードの保守性が高まることが期待できるであろう．このようなコードの整理の必要性を判断するための指標の一つとしても，*impCR* が利用できると考えている．

5.4 関 連 研 究

提案手法のように CVS リポジトリに登録されている情報を分析し，知見を得る研究は活発に行われている．例えば，German らによるソフトウェア変更の理由を把握するための研究 [7]，Herbsleb らによるソフトウェア開発におけるコミュニケーション遅延が開発にどのような影響を及ぼすかについての研究 [10]，Zimmermann らによる必要ではあるがなされていない潜在的な変更や不完全な変更を検知するための研究 [15] などが挙げられる．提案手法は，CVS リポジトリに登録されているソフトウェアの部品間の利用関係の変化を分析対象としたもので，リポジトリ内のソフトウェアの進化を新しい視点で分析したものであるといえる．

また，開発プロセスのおける活動やソフトウェアを分析する研究も活発に行われている．例えば，Johnson らは，開発者の活動を PC にインストールしたセンサを用いて記録し，内部的な特性（サイズや活動時間）と外部的な特性（プロダクトの質や信頼性）との関連を調査する研究 [13] を行っている．

また，大平らは，CVS などのコードリポジトリ，メーリングリストやバグ情報システムの履歴を蓄積して，それらの情報をグラフ化することで定量的なデータ収集とプロセス管理を行うシステム（EPM）を提案した [12]．このシステムは開発者や管理者間での情報共有がスムーズに行うことができるなど，有益なものであると思われるが，現状ソースコードに対する分析が LOC だけであるなど，ソースコードへの分析が十分でないと思われる．提案手法を EPM などのプロセス管理支援ツールに組み込むことで，より多彩な分析が可能になると思われる．

6. む す び

本論文では，ソフトウェア部品間の利用関係の変化をコンポーネントランクを用いて定量化することで，利用関係の変化という観点から，更新履歴中で重大な影響を与えた更新を抽出する手法を提案した．提案手法に基づきシステムを開発し，実際のオープンソースプロジェクトに適用したところ，大規模な機能追加以外に，コア部品に対する機能追加や利用関係の変化を伴うメンテナンス作業などが抽出できた．これらの更新の中には LOC の変化量が少ないものも多く，実験を通じて利用関係の変化をコンポーネントランクを用

いて定量化することが有益であることを確認した．

今後の課題として，メトリックスの精錬や，他の開発形態のプロジェクトへの適用などが考えられる．

謝辞 本研究は，科研費（19700033）及び 2007 年度南山大学パッヘ研究奨励金（I-A-2）の助成を受けている．

文　献

[1] 青山幹雄，中所武司，向山　博，コンポーネントウェア，共立出版，1998.
[2] 市井　誠，横森励士，松下　誠，井上克郎，“コンポーネントランクを用いたソフトウェアのクラス設計に関する分析手法の提案，” 信学技報，SS2005-37, 2005.
[3] 中塚　剛，松下　誠，井上克郎，“ソフトウェア部品分類手法へのコンポーネントランク法の応用，” 情処学研報，2007-SE-155, vol.2007, no.33, pp.41–48, 2007.
[4] 横森励士，梅森文彰，西　秀雄，山本哲男，松下　誠，楠本真二，井上克郎，“Java ソフトウェア部品検索システム SPARS-J，” 信学論（D-I），vol.J87-D-I, no.12, pp.1060–1068, Dec. 2004.
[5] 横森励士，藤原　晃，山本哲男，松下　誠，楠本真二，井上克郎，“利用実績に基づくソフトウェア部品重要度評価システム，” 信学論（D-I），vol.J86-D-I, no.9, pp.671–681, Sept. 2003.
[6] M.B. Chrissis, M. Konrad, and S. Shrum, CMMI: Guidelines for Process Integration and Product Improvement, Addison-Wesley Professional, 2003.
[7] D. German and A. Mockus, “Automating the measurement of open source projects，” Proc. 3rd Workshop on Open Source Software Engineering, pp.63–67, Portland, Oregon, 2003.
[8] G. Blom, L. Holst, and D. Sandell（著），森　真（訳），確率問題ゼミ，シュプリンガー・フェアラーク東京，1995.
[9] I. Jacobson, M. Griss, and P. Jonsson, Software Reuse, Addison Wesley, 1997.
[10] J.D. Herbsleb, A. Mockus, T.A. Finholt, and R.E. Grinter, “An empirical study of global software development: Distance and speed，” Proc. 23rd International Conference on Software Engineering, pp.81–90, Toronto, Canada, 2001.
[11] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, “Ranking significance of software components based on use relations，” Trans. Software Engineering, vol.31, no.3, pp.213–225, 2005.
[12] M. Ohira, R. Yokomori, M. Sakai, K. Matsumoto, K. Inoue, and K. Torii, “Empirical project monitor: A tool for mining multiple project data，” International Workshop on Mining Software Repositories (MSR2004), pp.42–46, Edinburgh, Scotland, 2004.
[13] P.M. Johnson, H. Kou, J.M. Agustin, Q. Zhang, A. Kagawa, and T. Yamashita, “Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from Hackystat- UH，” Proc. 2004 intl. Symposium on Empirical Software Engineering (ISESE2004), pp.136–144, Redondo beach, CA, 2004.
[14] R. Yokomori, M. Noro, and K. Inoue, “Evaluation of source code updates in software development based on component rank，” Proc. 13th Asia Pacific Software Engineering Conference (APSEC06), pp.327–334, Bangalore, India, Dec. 2006.
[15] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller, “Mining version histories to guide software changes，” Proc. 26th International Conference on Software Engineering, pp.563–572, Edinburgh, Scotland, 2004.
[16] “SourceForge，” http://sourceforge.net/
[17] “Galleon，” http://galleon.tv/
[18] “PMD，” http://pmd.sourceforge.net/
[19] “Ganttproject，” http://ganttproject.sourceforge.net/
[20] “JBidWatcher，” http://www.jbidwatcher.com/

（平成 19 年 6 月 29 日受付，10 月 19 日再受付）

横森　励士　（正員）

平 11 阪大・基礎工・情報卒．平 15 同大大学院博士課程了．同年同大学院情報科学研究科特任研究員．平 17 南山大・数理情報・講師．工博．プログラム解析を利用した開発支援に関する研究に従事．情報処理学会，IEEE 各会員．

野呂　昌満　（正員）

昭 56 慶大・工卒．昭 58 同大大学院博士前期課程了．昭 61 同大学院博士後期課程単位取得退学．同年南山大・経営・講師．助教授，教授を経て，平 12 同大・数理情報・教授．工博．ソフトウェアアーキテクチャ，アスペクト指向ソフトウェア開発に関する研究に従事．情報処理学会，日本ソフトウェア科学会，IEEE，ACM 各会員．

井上　克郎　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報・助手．昭 59〜61 ハワイ大マノア校・情報工学科・助教授．平元阪大・基礎工・情報・講師．平 3 同学科・助教授．平 7 同学科・教授．工学博士．平 14 阪大大学院情報科学研究科・教授．ソフトウェア工学の研究に従事．情報処理学会，日本ソフトウェア科学会，IEEE，ACM 各会員．