SArF: 依存関係に基づいてフィーチャーを集めるソフトウェアクラスタリング

小林健一[†1]　松尾昭彦[†1]　松下誠[†2]　井上克郎[†3]

概要：ソフトウェアを複数の小単位に分割するソフトウェアクラスタリング技術は，ソフトウェアシステムを理解する上で重要な役割を果たす．本論文では，静的な依存関係情報に基づいてソフトウェアのフィーチャーをクラスタに集めるソフトウェアクラスタリングアルゴリズム「SArF」を提案する．SArF の特徴はフィーチャーを集めることと，自動化である．ソフトウェアクラスタリング作業の多くは，遍在モジュールの除去作業など人間の支援を必要とするが，SArF はこれら人間の作業が不要である．SArF の特徴を実現するために，依存関係に対してフィーチャーを共有する確からしさを表す専念度(Dedication)スコアを定義し，スコアで重み付けられた依存関係グラフをクラスタリングするためにモジュラリティ最大化法と組み合わせた．ケーススタディにてフィーチャーが集められることを示し，公開リポジトリから利用度順に収集した 35 ソフトウェア 304 バージョンからなるデータセットを用いて SArF の評価を行い，クラスタリング品質，安定度，実行時間の面で SArF が既存のアルゴリズムに対し優れることを示した．

キーワード：ソフトウェアクラスタリング，ソフトウェアアーキテクチャ復元，プログラム理解，依存解析

SArF: Feature-Gathering Dependency-Based Software Clustering

KENICHI KOBAYASHI[†1]　AKIHIKO MATSUO[†1]　MAKOTO MATSUSHITA[†2]
KATSURO INOUE[†3]

Abstract: Software clustering that decomposes a software system into plural sub-units is one of the important techniques to comprehend software systems. In this paper, we proposed a novel dependency-based software clustering algorithm, SArF. SArF has two characteristics. First, SArF gathers relevant software features or functionalities into a cluster. Second, SArF further automates software comprehension processes by eliminating the need of the omnipresent-module-removing step which requires human interactions. To achieve them, we defined the Dedication score to measure the likelihood that a dependency shares the same features, and we utilized modularity maximization method to cluster directed graphs weighted by the Dedication score. A case study shows that SArF could successfully gather relevant features into a cluster. Extensive comparative evaluations using 35 popular open-source software systems (total 304 versions) show that SArF is superior to existing dependency-based software clustering studies in terms of clustering quality, stability and speed.

Keywords: Software clustering, Software architecture reconstruction, Program comprehension, Dependency analysis

1. はじめに

ソフトウェアシステムは長年の保守や機能追加を経るにつれてドキュメントの陳腐化や知識の散逸が起こる．そのため，アーキテクチャ理解はソフトウェア保守の重要な位置を占める[1]．ソフトウェアクラスタリングはソフトウェアの構成要素であるモジュール（ソースファイルやクラス，メソッド，データエンティティ等）をクラスタリングすることで，ソフトウェアを複数の小単位に分割してその潜在構造を明らかにするアーキテクチャ理解のための技術である．ソフトウェアクラスタリングによる分割結果は，対象ソフトウェアのアーキテクチャ知識や高レベルな抽象ビューとして利用できる．

ソフトウェアクラスタリングの用途の一つは，過去にドキュメント化されたアーキテクチャと現状のアーキテクチャの乖離を把握し是正することである．例えば，設計当初のパッケージ構造（本論文での「パッケージ」は Java 言語のようなモジュールの集合体を指す）をそのままに，安易にクラスを追加する拡張を続けた場合，パッケージ構造と実態が乖離していく．ソフトウェアクラスタリングの出力は実態を映すビューであり，パッケージ構造を実態に合わせるリファクタリングを可能にする．別の用途としては，ドキュメント化されていない観点のビューを新たな知識として加えることが挙げられる．アーキテクチャの観点は複数あるため[1]，開発者が求める観点とは異なるパッケージ構造が採られることがしばしばある．例えば，ウェブフレームワークによってアーキテクチャが規定され，それに合わせてパッケージ構造が強制される場合などである．このような場合，パッケージ構造から得られない非自明な知識が抽出できるソフトウェアクラスタリングが有用である．

現在までに，様々な目的を持ったソフトウェアクラスタリングの研究が多く行われており，各々が目的に沿ったアーキテクチャビューを提供する．例えば，モジュール性を高めるように構造を見直す目的には，クラスタ内の凝集性

†1 富士通株式会社（第 2 著者の所属は本研究実施当時のもの）
Fujitsu Limited.
†2 大阪大学大学院情報工学研究科
Graduate School of Information Science and Technology, Osaka University
†3 南山大学理工学部ソフトウェア工学科
Faculty of Science and Technology, Nanzan University

が高くクラスタ間の結合性が低い集合を探索するアルゴリズム[2]が向いている．このように，開発者は自らの目的に沿ったアルゴリズムを選ぶことができる．

我々は依存関係に基づく新しいソフトウェアクラスタリングアルゴリズム「SArF」（**S**oftware **Ar**chitecture **F**inder の略）を提案する．SArF の目的はフィーチャーを実装するモジュールを同じクラスタに集めることである．本研究では「フィーチャー」をフィーチャーロケーション（機能捜索）の研究の文脈での定義，すなわち，「システム外のユーザから引き起こされうる観測可能な機能（つまり，非機能要件を含まない）」[3]という意味で用いる．例えばデータ処理でのフィルタリング機能や，GUI でのダイアログ機能などである．フィーチャーを集めることで，既存手法では得られなかった機能に基づいた分割結果のビューを得ることができる．出力結果の各クラスタに対して，どのようなフィーチャーであるかの解釈や，フィーチャーに包含関係や重複関係がある場合の対応付けなどは，結果を利用する開発者に委ねられる．

SArF はソフトウェアクラスタリングを高いレベルで自動化していることも特徴である. SArF は適用可能性を高めるために収集が容易な静的依存関係（呼出関係, 参照関係, 継承関係など）のみを入力情報として用いている．依存情報を用いるソフトウェアクラスタリングの既存のアプローチ[2][4][5][6]では，利用者が満足のいくまで精錬された結果を得るためには，分割結果を人間が分析してフィードバックし反復するという介入が必要である[1][7][8]．介入を要する大きな要因が「遍在モジュール(omnipresent module)」[9]の存在である．遍在モジュールとは，システムの複数個所と接続しながら，特定の 1 サブシステムには属さないモジュールと定義される[2]．遍在モジュールは依存情報においてノイズとして振る舞うため[9]，多くの研究が結果を精錬するために遍在モジュールを除去することが有益であるとしている[2][10][11][12]．遍在モジュールの自動除去についてはいくつかの提案[4][9][13][14]があるが，遍在モジュールの除去の実施判断や自動除去ツールのパラメータは人間が決定する必要があり，さらに遍在モジュール除去の妥当性の確認は手動で行う必要があるなど，実際には半自動化に留まっていた. SArF は遍在モジュールの存在に影響を受けにくいよう設計されたアルゴリズムであり，遍在モジュールを他のモジュールとアルゴリズム上で同等に扱えるようにすることで，除去などの特別な措置が不要になり，結果，クラスタリングに際して人手の除去作業やパラメータ調整が無くなり自動化が達成される．

SArF のフィーチャーを集める特徴と, 遍在モジュール除去作業を不要にする特徴は，依存関係の重要度を表す専念度(Dedication)スコアを定義し，専念度スコアで重み付けられた依存グラフをコミュニティ発見の研究分野で用いられるモジュラリティ最大化法[15]を用いることにより実現された．

SArF がフィーチャーを集めることは，[16]にて開発へのインタビューを含む実例で示し，複数の既存研究と比較して SArF が人間の介入なしで高い精度を出すことを示した．本論文では妥当性を増すため，客観的な選択規準で十分な数のソフトウェアを公開リポジトリから取得して実験評価を行った．また，[16]ではモジュラリティ最大化法のアルゴリズムとして Newman アルゴリズム[15][17]を用いていたが，より優れるとされる Louvain 法[18]に置き換えた．単なる置き換えでは安定度の低下を招いたが，Louvain 法のアルゴリズムに修正を加えることで安定度を同等に保つことができ，クラスタリング品質の向上と実行時間の短縮が実現された．

本論文の以降の構成は次の通りである．2 章にて関連研究を示し，3 章にてソフトウェアクラスタリングアルゴリズム SArF の提案を行い，4 章では実験計画について述べ，5 章でケーススタディを示し，6 章で実験結果を示し，議論する. 7 章で妥当性への脅威を示し, 8 章にて結論を述べる．

2. 関連研究

2.1 ソフトウェアクラスタリングアルゴリズム

ソフトウェアクラスタリングの既存研究を，まず入力情報の観点で述べると，依存情報や構造情報が用いられることが多い．Bunch [2]は最適化によるグラフクラスタリングアプローチであり，内部に高い凝集度を持ち外部と低い結合度を持つクラスタを発見することを目的とする．ACDC [4]はグラフパターンを用いるアプローチであり，クラスタリングのために「グラフの支配ノードと被支配ノードのモジュールをまとめる」など, 複数の発見的ルールを用いる．関連する名前を持つモジュールをまとめるために命名規則を利用する手法もある[19]．自然言語処理により，ソースコード中の識別子やコメントなどの意味的情報を用いる手法もある[20][21]．システムの振る舞いを理解するために，実行トレースなどの動的情報を用いてアーキテクチャを復元する手法[22]やクラスタリングを行う手法[12]もある．

次に，方法論の観点で述べると，階層クラスタリングを用いる手法[10][21][23]，k-means などの非階層クラスタリングを用いる手法[20]，パターンマッチングを用いる手法[4], グラフクラスタリングを用いる手法[2][5][7][24]などがある．Bittencourt らは 4 つのグラフクラスタリング手法の比較を行っている[24]．グラフクラスタリングは辺の密度が高いサブグラフを見つけるクラスタリングであり，辺として依存関係のような直接的な情報を用いるとクラスタの解釈が容易で結果を直接利用できる．特に初期に提案された Bunch は多くの改良が提案され，コンポーネントの再利用性の指標を遍在モジュール除去に用いるもの[14]，ユーザの目的により適合させるため多目的最適化を用いるもの[7]，対話的遺伝アルゴリズムを用いるもの[8]などがある．

近年では，生物学的ネットワークやソーシャルネットワークの応用において，コミュニティ発見のグラフクラスタリング技術が急速に発展している．Girvan-Newman (GN)アルゴリズム[25]は，高い Edge Betweenness 指標値を持つ辺を切断することによってグラフクラスタリングをトップダウン的に行うアプローチである．GN アルゴリズムは小さなソフトウェアでは良好な性能が得られたが，大きなソフトウェアでは性能が低下したという評価が報告されている[5][24]．モジュラリティ最大化法の発端となった Newman アルゴリズム[15]は，グラフのノードやクラスタを併合するボトムアップアプローチであり，良好な性能が得られたと報告されている[6].

最後に，目的の観点で述べると，各クラスタリング手法はその入力情報（依存情報・構造情報，意味的情報，動的情報）で大まかに目的が分かれ，各々が採用した方法論によって特徴づけられる．例えば，モジュール性を高める目的には依存情報を用いて凝集度を最大化する手法が向く，などである．SArF は，依存情報を用いたグラフクラスタリングという従来の枠組みに加えて，専念度という評価指標を導入して依存関係に重みづけることにより，フィーチャーを集めるという新しい目的を持つことが特徴である．

2.2 ソフトウェアクラスタリングの評価

ソフトウェアクラスタリングアルゴリズムを評価するために一般的に用いられる規準はオーソリティ準拠度（Authoritativeness）である．これは，アルゴリズムが出力した分割結果と，対象ソフトウェアのオーソリティ（権威者, 熟知者）が作成したオーソリティ分割結果（Authoritative Decomposition）との間の類似度または距離として定義される．MoJo[26]はオーソリティ準拠度の指標で，二つの分割を一致させるために必要な最小の move 操作と join 操作（図 3）の合計数である．MoJo の欠点を改善した指標 MoJoFM が提案され[27]，階層的性質を持つ分割を評価するため，up 操作を考慮する指標 UpMoJo [28]が提案されている．

一つのソフトウェアには複数の観点がありえるため，正解となる分割結果も複数個共存しえる．よって，アルゴリズムが一つのオーソリティ分割結果にはそれほど準拠しなくても，他のオーソリティ分割結果には準拠する可能性がある．Shtern と Tzerpos は目的が異なるクラスタリングアルゴリズムは直接的には比較できない場合があることを指摘し[29], 複数の指標を用いた評価の枠組みを提案した[30].

Wu ら[23]は，ソフトウェアクラスタリングが備えるべき性質の規準, オーソリティ準拠度・NED・安定度（Stability）を設定し, 複数のクラスタリングアルゴリズムを比較した．NED は極端な結果を出力しないことの指標であるが, 定義が恣意的と指摘されている[20]．安定度は小さな変更に対して結果が大きく変化しない性質の指標である．Lutellier らは MoJoFM など 4 つのアーキテクチャ比較指標を用いて[2][4][11][21]など 9 つの手法の比較を行った[31]．本研究の評価はこれら既存の枠組みに則って行う．

3. 手法

本章では提案手法 SArF アルゴリズムについて述べる．SArF アルゴリズムは, 依存関係の辺を重み付けする専念度スコアと，その重み付き依存グラフをモジュラリティ最大化法でクラスタリングすることの 2 点からなる．

3.1 専念度(Dedication)スコア

専念度スコアは「依存するモジュールと依存されるモジュールが同じフィーチャーを共有する確からしさ」を表す, 依存関係グラフの辺に与える重みである．これは，依存されるモジュールが依存しているモジュールのフィーチャーの実装にどれほど専念しているかを意味している. 例えば, 図 1 左はモジュール A がモジュール X に依存していることを表しているが, もし X が A にのみ専念しているならば, X は A と同じフィーチャーを共有している可能性が高い．一方，図 1 右のように，多くのモジュールがモジュール Y に依存しているならば，Y は特定のモジュールに専念しているとは言い難く，それらとフィーチャーを共有している可能性は低い．

上記の考え方に基づく専念度スコアの定義を述べる．依存関係の集合を有向グラフとして扱うものとして，頂点はモジュール（ソースファイル，クラス，メソッド，データエンティティなど）を表す．A が B に依存するとき，有向辺(A,B)が存在するとする．単純な場合には，辺(A, B)の専念度スコア D(A,B)の定義は，fanin(B)を頂点 B の入次数として，次式となる．

$$\mathrm{D}(A,B)=\frac{1}{\mathrm{fanin}(B)} \tag{1}$$

図 1 の例の各専念度は D(A,X) = 1, D(B_1,Y) = 1/50 となる．遍在モジュールはこの Y の形態をとるため，遍在モジュールへの専念度スコアは常に小さい．

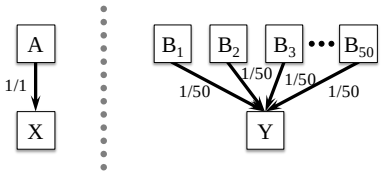


図 1　専念度スコアの計算例（単純な場合）

Figure 1　Example of Dedication score (simple case)

クラスとメンバといった階層関係が利用できる場合には，メンバよりクラスをモジュールの単位としてクラスタリングを行う方が解釈や管理が容易である．メンバレベルの依存関係グラフの情報を集約してクラスレベルとする場合の専念度スコア $\mathrm{D_M}$(A,B)の定義は次式となる．

$$\mathrm{D_M}(A,B)=\sum_{m\in M_{B\leftarrow A}}\frac{1}{\mathrm{fanin_X}(m)\cdot \mathrm{m_X}(B)} \tag{2}$$

ここで，A と B はクラス，$M_{B\leftarrow A}$ は B のメンバのうち A のメンバが依存するものの集合である．$\mathrm{fanin}_X(m)$はメンバ m が属するクラス以外から m に依存するメンバ数である．$m_X(B)$は B のメンバのうち B の外から依存されるものの数であり，クラス全体の専念度を分け合う数となる．クラスレベルグラフの専念度スコアは対応するメンバレベルグラフと式(2)を用いて計算される．式(2)の例を図 2 に示す．$m_X(X)=3$，$\mathrm{fanin}_X(X.init)=1$，$\mathrm{fanin}_X(X.set)=1$，$\mathrm{fanin}_X(X.use)=3$ である．クラス A からクラス X への専念度 $D_M(A,X)=7/9$ のうち，メソッド A.a からメソッド X.init への分は 1/3，A.a から X.set への分は 1/3，A.f から X.use への分は 1/9 である．

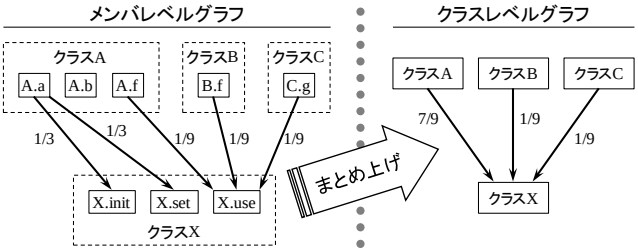


図 2　専念度スコアの計算例（階層関係がある場合）

Figure 2　Example of Dedication score (multi-level case)

3.2 モジュラリティ最大化法

専念度スコアが効果を発揮するクラスタリングアルゴリズムを選択する場合，その要件は，専念度スコアの高い依存関係をクラスタ内に有し，かつ，遍在モジュールのような専念度スコアの低いノイズとなる依存関係に影響を受けないことである．ここで，有意な依存関係の規準として，専念度スコアがその期待値を上回る度合を用いる．それに適する手法としてモジュラリティ最大化法[15]を採用する．

モジュラリティ最大化法はコミュニティ発見のアプローチの一つであり，目的関数モジュラリティ Q [32]を最大化するグラフの分割を求めるものである．モジュラリティ Q は重み付きグラフに自然に拡張され，後に次式の通りモジュラリティ Q_D として有向グラフに拡張された[33][34].

$$Q_D = \frac{1}{W}\sum_{i,j}\left[A_{ij} - \frac{k_i^{OUT} k_j^{IN}}{W}\right]\delta(c_i, c_j) \qquad (3)$$

ここで，W はグラフ内のすべての有向辺の重みの合計であり，A_{ij} はグラフの隣接行列の要素すなわち辺(i,j)の重みであり，k_i^{OUT} は頂点 i から出る辺の重みの合計であり，k_j^{IN} は頂点 j へ入る辺の重みの合計であり，c_x は頂点 x が属するクラスタであり，$\delta(c_i,c_j)$は $c_i = c_j$ の場合に 1 となりそれ以外の場合に 0 となるクロネッカーのデルタ関数である．式(3)中の項$(k_i^{OUT}\ k_j^{IN}\ /\ W)$は辺$(i,j)$の重みの期待値であり，項 A_{ij} は実際の重みである．専念度スコアが辺の重みとなるので，式(3)の括弧の中の式は前述の規準に一致している．

モジュラリティ Q_D はクラスタ内の辺がその期待値よりどれだけ高い密度を持っているかの程度と解釈され，高い値ほど良いクラスタリング結果を意味する．モジュラリティ Q の最適化は NP 困難である[35]が，良い近似を与える解法が存在している．我々は[16]で Newman のアルゴリズム[15]を用いた．その計算量は，|V|をグラフの頂点数とすると，典型的に $O(|V|\log^2|V|)$と高速である[17]．本論文では Louvain 法[18]を用いる．Louvain 法は非常に高速（典型的に $O(|V|\log|V|)$），かつ，貪欲法でありながら他の計算量の大きい手法と同等以上に良質の解を得ることができる[36]．以降，「SArF」は Louvain 法を用いた SArF を表す．比較のため，Newman アルゴリズムを用いた以前の SArF は「SArF(N)」と表す．

以下に，Louvain 法の手順概略を示す（詳細は[18]）．

1. 各頂点を 1 つだけ含むクラスタを頂点数と同数作る．
2. すべての頂点を順に走査し，各頂点を Q_D の増加が最大になるクラスタへ移動する．
3. 2.の走査を Q_D が増加しなくなるまで繰り返す．
4. クラスタを頂点とする新しいグラフを作り，Q_D が増加する限り，1.に戻る．

Louvain 法はそのままソフトウェアクラスタリングに適用するのは不適である．ソフトウェアクラスタリングでは結果は決定的かつ安定度が高いことが望まれるのだが，オリジナルの Louvain 法は手順 2.の走査が乱数順であるため実行ごとに結果が非決定的であり，安定度も若干低くなる．SArF では決定的な結果を得るために Louvain 法の走査順を，(1)重み付き次数の昇順に従い，(2)同位の場合は隣接頂点の重み付き次数の合計値の昇順に従う，とする固定順に変更する．走査順をグラフの構造に基づいて定めることで，似たアーキテクチャでは似た走査順となり安定度が高まる効果が得られる．この固定順走査への変更の影響について 6.3.6 節で調べる．

3.3 SArF アルゴリズム

我々が提案する SArF アルゴリズムは専念度スコアとモジュラリティ最大化法の組み合わせである．このアルゴリズムは決定的であり，同じ入力に対して同じ結果を出力する．このアルゴリズムの手順は以下の通りである．

1. モジュール（クラス，メンバ，その他のエンティティ）間の依存関係を対象のソフトウェアから抽出する．
2. 抽出した情報から依存関係グラフを作成する．可能ならメンバレベルのグラフを抽出する．
3. 3.1 節で述べた手順で，依存関係グラフの専念度スコアを計算する．グラフがメンバレベルならば，クラスレベルグラフにまとめ上げる．
4. 3.2 節で述べた手順で，重み付き有向のモジュラリティ最大化法によりグラフのクラスタリングを行う．

4. 実験計画

本章ではリサーチクエスチョンを設定し，それに答えるために必要な性能指標と性能測定の方法について説明する．

4.1 リサーチクエスチョン

本論文では以下のリサーチクエスチョンを設ける．

- **RQ1**「SArF はフィーチャーを集められるか?」
- **RQ2**「SArF はソフトウェアクラスタリングをより自動化できたか?」
- **RQ3**「SArF はソフトウェアクラスタリングにおいてよい特徴を備えているか?」

RQ1 と RQ2 を検証するために，フィーチャーに基づく観点からソフトウェアを精査するケーススタディを 5 章で行う．また，RQ1 と RQ2，RQ3 を検証するために，6 章では多数の OSS を対象とした実験評価を行う．リサーチクエスチョンの検証を 6.4 節でまとめる．

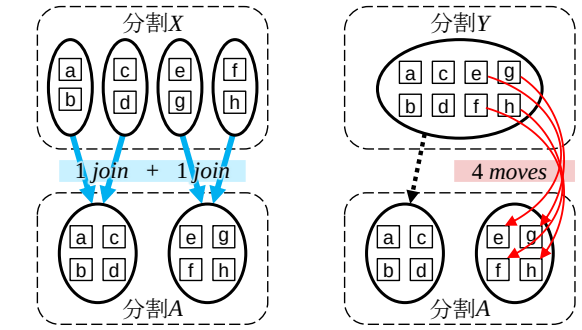
4.2 性能指標

性能評価の軸として，「品質」「安定度」「実行時間」の 3 軸を設定し，「品質」をさらに「オーソリティ準拠度」「適切なクラスタ数」の 2 つの規準で測る．広く用いられる Wu らの評価の枠組み[23]を，批判[20]のある NED を「適切なクラスタ数」に換える形で用いた．

オーソリティ準拠度の算出に必要となるオーソリティ分割結果の獲得は膨大な労力を要すため，既存研究のほとんどが対象ソフトウェアのパッケージ階層をオーソリティ分割結果の代替として利用している．我々も[20][21]と同様に，各パッケージをクラスタとし，サイズ 5 以下のクラスタを再帰的に親クラスタに併合することにより，パッケージ階層からオーソリティ分割結果の代替物を作成した．

ここで注意すべき点は，パッケージ階層から作成したオーソリティ分割結果は，そのパッケージ階層の設計がどの観点によって行われたかに依存するということである．例えば, SArF の評価にはフィーチャーの観点で分割されたオーソリティ分割結果が望ましいため，5 章のケーススタディではそのようなオーソリティ分割結果が得られるソフトウェアを選んでいる．一方，6 章では無作為にソフトウェアを選ぶ大規模な評価を行っており，この場合は特定の観点に寄ることのない任意かつ多様な観点を寄せ集めた「平均的な基準」による評価となる．このような「平均的な基準」により複数のソフトウェアクラスタリング手法を比較しなければならないのは，現状のソフトウェアクラスタリングの研究の共通課題である．我々は評価の妥当性を増すために，「平均的な基準」による評価と，自身の観点（フィーチャー）に沿ったケーススタディを併せた二段構成の評価を実施している．

オーソリティ準拠度の指標としては MoJoFM[27]を用いた．MoJoFM は 2 つの分割結果の類似度であり，MoJoFM(C,A) = (1 – mno(C,A) / N_{maxops})×100% で定義される．C は出力分割結果, A はオーソリティ分割結果, mno(C,A) は C を A へ変換するのに最小限必要な move 操作と join 操作の合計数，N_{maxops} は最大操作数（= max$_X$ mno(X,A)）である．高い MoJoFM は C が A に近いことを意味する．例を図 3 に示す．



MoJoFM(X,A) = 1 – 2 / 8 = 75%　　MoJoFM(Y,A) = 1 – 4 / 8 = 50%

図 3　move／join 操作と MoJoFM

Figure 3　Move / join operations and MoJoFM

MoJoFM はクラスタ数が過多の場合に過大評価を起こすバイアスがある．図 3 の例から判るように，クラスタを誤って 2 分した場合 join 操作 1 回分のペナルティを受けるのに対し，誤って 2 つのクラスタを併合した場合には多数の move 操作分のペナルティを受ける．ペナルティに対称性がないため，過多分割の傾向を持つアルゴリズムが高く評価される．よって，バイアス補正のため，クラスタ数が適切であるかの規準を加える必要がある．

適切なクラスタ数の指標として，オーソリティ分割結果を規準としたクラスタ数の相対誤差(RE, Relative Error)と絶対相対誤差(MRE, Magnitude of RE)を用いる．本論文を通し，K はアルゴリズムが出力した分割結果のクラスタ数，Ka はオーソリティ分割結果のクラスタ数を表す．MRE は RE の絶対値，RE は RE = (K – Ka) / Ka と定義される．MRE が小さいほど，オーソリティ分割結果からみて適切なクラスタ数やクラスタ粒度と言える．

安定度は，対象ソフトウェアの第 i バージョンの分割結果を C_i とし, その前のバージョンの分割を C_{i-1} としたとき，Stability(C_i) = MoJoQ(C_{i-1},C_i)で定義される．MoJoQ[26]はモジュール数が N のとき，MoJoQ(C,A) = (1 – min(mno(C,A), mno(A,C)) / N)×100% と定義され，MoJoFM と異なり対称な類似度である．

実行時間は，図 4 に示された期間の経過時間である．すなわち, SArF では専念度スコアの計算とクラスタリングの合計実行時間を測り，他のアルゴリズムについてはクラスタリングの実行時間のみを測る．

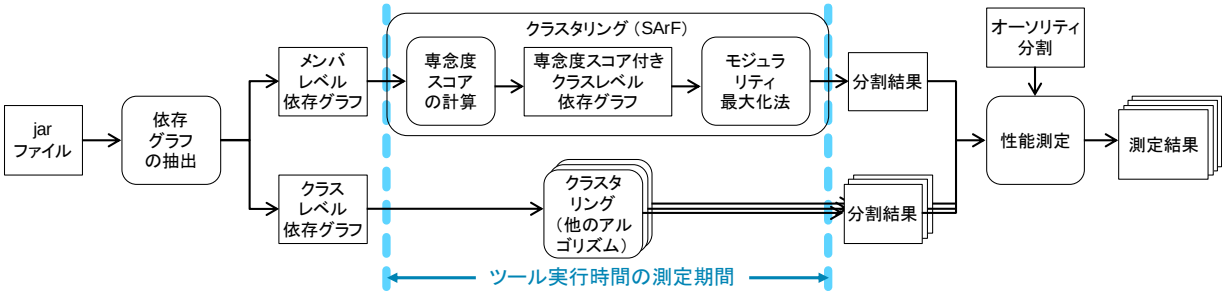


図 4　性能測定手順

Figure 4　Measurement Procedure

4.3 性能測定手順

性能測定手順を図 4 に示す．評価対象ソフトウェアはすべて Java で記述されており，入力情報は jar ファイルのみを使用する．jar ファイル中の対象ソフトウェアに属さないクラス（例：ant ならば `org.apache.tools.zip` パッケージ以下）は評価対象外とする．まず，jar ファイルから Javassist[*1] を基に開発されたバイトコード解析器を用いてメンバレベルとクラスレベルの依存関係グラフを抽出する．抽出する依存関係はメソッド呼出・フィールドリード・フィールドライト・継承・クラス型参照の 5 種であり，これは高いクラスタリング品質を得る上で十分な組合せである[37]．単一のソースファイルにクラスが複数含まれる場合，それらが同じグループに属すことは自明なため，すべて最初のトップレベルクラスに属するものとみなす．クラスレベルグラフはメンバレベルグラフを図 2 のようにまとめ上げることで作成する．依存関係を持たないクラスは評価対象外とする．オーソリティ分割結果はソフトウェアのパッケージ階層から生成されるため，評価の公正を保つため，パッケージ情報は依存関係グラフから除去しておく．次に，各クラスタリングアルゴリズムを実行する．比較評価で用いたすべてのクラスタリングツールは既定値のパラメータにて実行する．最後に，出力された分割に対し測定された前述の性能指標を収集する．手順中は実行時間も測定する．

5. ケーススタディ

本章では，RQ1「SArF はフィーチャーを集められるか?」の検証を主目的とし，2 つのソフトウェアを用いてケーススタディを実施する．先述の通り，フィーチャーを集めるソフトウェアクラスタリングの検証にはフィーチャーの観点に基づくオーソリティ分割結果が必要となる．そのため，本ケーススタディでは既存のアーキテクチャ知識を用いてオーソリティ分割結果を作成した．また，1 番目のケーススタディでは，RQ2「SArF はソフトウェアクラスタリングをより自動化できたか?」の検証のために，遍在モジュールがクラスタリング結果に及ぼす結果についても調べる．

5.1 ケーススタディ 1：Weka 3.0

本節では，SArF がフィーチャーを集めることができるかどうかについて直観的理解を得るため，データマイニングツール Weka のバージョン 3.0 を題材として可視化を用いたケーススタディを示す．Weka 3.0 はアーキテクチャが文書化されている[38]ため，既存研究でよく用いられる[6][12]．そのほとんどのパッケージがフィーチャーに一致する[12]ため，パッケージを元にしたオーソリティ分割結果は SArF の目的に合致する．このケーススタディの目的は，SArF と他のクラスタリングアルゴリズムの結果を比較し，SArF がフィーチャーと一致する分割を出力することの実例を示すことである．[16]により詳細な解析がある．

ここでは Weka バージョン 3.0.6 の `weka` パッケージ以下すべて 142 個のクラスを対象とした．図 5 は，文献[12][38]と jar ファイルの情報を利用して作成した Weka のアーキテクチャを表すパッケージダイアグラムである．`core` 以外のパッケージは Weka のフィーチャーに対応する．

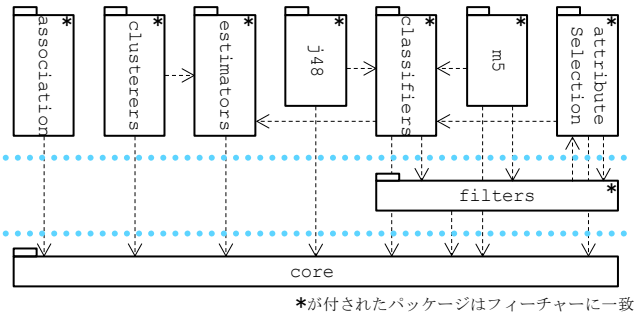


図 5　Weka 3.0 のアーキテクチャ

Figure 5　Architecture of Weka 3.0

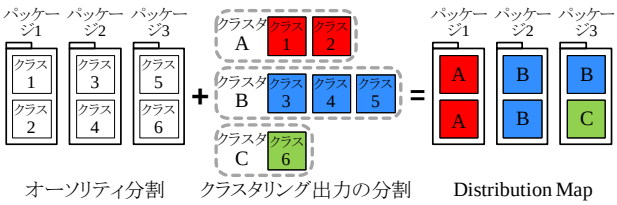


図 6　Distribution Map による可視化の例

Figure 6　Example of Distribution Map

クラスタリング結果をオーソリティ分割結果との差異として可視化するため図 6 に例を示す Distribution Map[39]の技法を用いる．オーソリティ分割結果はパッケージダイアグラムで表し，クラスタリング結果は各クラスの色とア

*1 `http://www.javassist.org/`

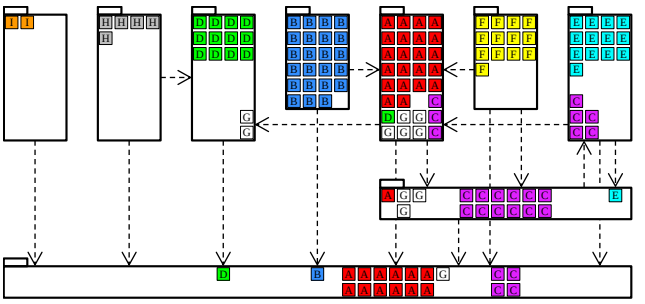
ルファベットでクラスタを識別することで表す．オーソリティ分割結果とクラスタリング出力が一致するならば，パッケージは個別の色で占められ（パッケージ 1），異なるならばパッケージ内に混色が生じたり（パッケージ 3），同じ色（クラスタ B）が複数のパッケージにまたがったりする．

4.3 節の手順に従ってクラスタリングアルゴリズムを実行した結果を可視化した例を図 7 に示す．図 7(a)は SArF の結果である．各クラスタが `core` 以外のパッケージにほぼ一致していることが判る．これは，SArF が効果的にクラスを対応するフィーチャーに分割していることを意味し，MoJoFM 値が 72.9%と高いことに整合する．なお，最も正確な結果が得られた場合であっても，一般的にオーソリティ準拠度は 100%より低い値になる．なぜなら，オーソリティ準拠度が 100%となるためは，指標の基準となるオーソリティ分割結果も理想的なものが与えられる必要があるが，現実にはそのようなオーソリティ分割結果が得られることはまず無いためである．すなわち，基準とするオーソリティ分割結果ごとに上限値が存在することになる[16].

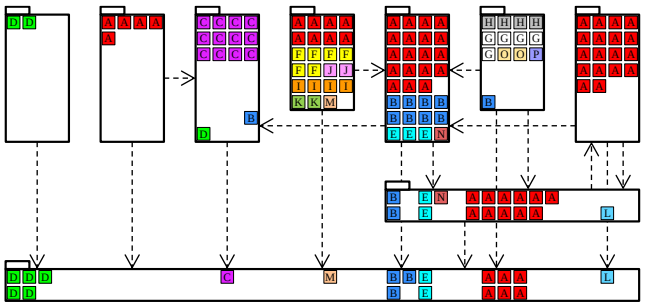
図 7(a)には，クラスタの一部がパッケージと一致していない箇所がある．例えば，白「G」クラスタは複数のパッケージに散逸している．この誤分類の原因は大きく二つある．一つ目は，複数のフィーチャーに関わるクラスがあるときに，どちらに分類されるかが，パッケージ設計者のそれと，SArF のそれで一致しない場合である．二つ目は，フィーチャーの中に小さなフィーチャーが入れ子になっている場合である．この例では，`classifier` パッケージは「決定木以外の判別分析（回帰分析も含む）」というフィーチャーに対応するが，SArF はその中から「回帰分析」というフィーチャーを白「G」クラスタとして取り出している．

比較対象アルゴリズム ACDC と Bunch の結果を図 7(b)と図 7(c)に示す．また，SArF の結果と併せて表 1 第 1 列に示す．「*Ka*」列はオーソリティ分割結果のクラスタ数を表す．「*K*」列は各アルゴリズムが出力した分割のクラスタ数を表す．アルゴリズム間で最良の MoJoFM 値を太字で表す．ACDC と Bunch に共通して，少数のクラスタ（赤「A」と青「B」）が `filters` パッケージと `core` パッケージを介して複数のパッケージにまたがっている．これから両パッケージ中の遍在モジュールが，クラスタリングアルゴリズムが正しく働くことを阻んでいることが観察できる．これら既存のアルゴリズムでは，望ましい結果を得るために遍在モジュールを除去する人間の介入が必要となっていた．ACDC の結果には非常に多く（クラスタ数 16）の小さなクラスタが散らばっており，一方 Bunch の結果では複数のパッケージが一つのクラスタに融合しており，両者でクラスタ粒度の不整合が起きていることが判る．

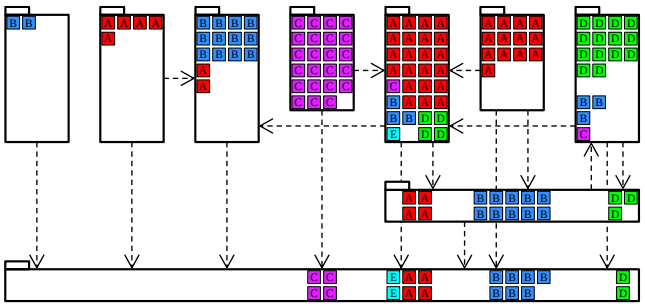
次に，遍在モジュール除去が結果に及ぼす効果を調べる．表 1 第 2 行「Weka 3.0.6 (feature, OM 除去)」は，「Weka 3.0.6」から遍在モジュールを除去したものを対象としたクラスタ



(a) SArFの結果（クラスタ数:9，MoJoFM: 72.9%）



(b) ACDCの結果（クラスタ数:16，MoJoFM: 47.4%）



(c) Bunchの結果（クラスタ数:5，MoJoFM: 32.3%）

図 7　Weka のクラスタリング結果の可視化
Figure 7　Visualization of Clustering Results on Weka

リング結果である．遍在モジュール除去のために，既存研究[6][12]と同じくクラスタリング前に依存関係グラフから `filters` パッケージと `core` パッケージをすべて取り除いた．比較のため，表 1 第 3 行「Weka 3.0.6 (feature)」として，第 1 行と同じ「Weka 3.0.6」を対象としたクラスタリング結果に対し，`filters` パッケージと `core` パッケージを除いて MoJoFM 値を計算したものを用意した（残ったパッケージはすべてフィーチャーに基づくパッケージ）．第 2 行と第 3 行を比較することで，遍在モジュールによってクラスタリングが阻害される様を観察できる．まず，SArF は遍在モジュール除去を行った場合（第 2 行）と行わなかった場合（第 3 行）の MoJoFM 値が 97.0%と 90.9%であり差は少なく，両方とも高い性能が得られている．他方，ACDC では除去ありで 85.9%に対し，除去なしで 55.6%であり，遍在モジュール除去の効果が大きいことが判る．Bunch では除去ありで 70.1%に対し，除去なしで 59.6%であり，遍在

モジュール除去の効果はあることが認められるが，除去してもよい性能は得られていない．ただし，この比較ではBunch の性能が悪いのか，フィーチャーに基づく評価に向かないのかの二者を区別することはできない．

表 1　Weka 3.0 における性能測定結果

Table 1　Performance Measurement Results on Weka 3.0

ソフトウェア名	クラス数	*Ka*	SArF		ACDC		Bunch*	
			MoJoFM	*K*	MoJoFM	*K*	MoJoFM	*K*
Weka 3.0.6	142	9	**72.9**	9	47.4	16	44.1	3-7
Weka 3.0.6 (feature, OM 除去)	106	7	**97.0**	9	85.9	5	70.1	5-9
Weka 3.0.6 (feature)		7	**90.9**	9	55.6	15	59.6	3-7

*Bunch の結果は乱数による影響が大きく現れ，評価値の分散が大きいため，5 回平均の結果を記す．他の表でも*付きの列は同様

5.2 ケーススタディ 2 : 商用データマイニングツール

本ケーススタディでは，フィーチャーに基づくオーソリティ分割を, 対象ソフトウェアの有知識者によって作成し, SArF の効果を調べることを目的とする. 対象ソフトウェアとして富士通で開発されたデータマイニング製品（DMTool と仮称）を用い, 2 つのオーソリティ分割結果を用意した．第 1 の分割 ADpackage はパッケージ階層から自動生成したものであり，第 2 の分割 ADfeature はこの製品の開発者がフィーチャーの観点で作成したものである．

このソフトウェアはウェブ 3 層アーキテクチャを採っており，パッケージの設計もそれに沿っているため，ADpackage と ADfeature は異なる観点に基づく分割となっている．アーキテクチャと 2 つの分割についてのこれ以上の詳細はここでは割愛するが，[16]にその詳細が述べられ，ケーススタディ 1 と同様の Distribution Map 技法を用いた可視化が示されている．

この 2 つの分割を用いた性能評価結果を表 2 に示す（各列の意味は表 1 に同じ）．第 1 行(ADpackage)はクラスタリング結果に対し ADpackage をオーソリティ分割として評価したものであり，第 2 行(ADfeature)は同じクラスタリング結果に対し ADfeature をオーソリティ分割として評価したものである．

表の第 1 行と第 2 行を比較すると，ACDC と Bunch は大きな性能差は無いが，SArF は ADfeature において大幅な性能増加があることが判る．また，SArF は他のアルゴリズムに比べて性能が高い．これは，SArF の出力結果が開発者の作成したフィーチャーに基づくオーソリティ分割に近いことを意味し，RQ1「SArF はフィーチャーを集められるか?」を肯定的に支持する．また，これらの事実はオーソリティ分割がどの観点に基づくかによって，クラスタリングアルゴリズムの性能評価値に差が出るという従来からの知見[29]を補強する．

SArF のクラスタリング結果の妥当性を確認するために，DMTool の開発者がクラスタリング結果と自身のソフトウェア知識と照合したところ, SArF のクラスタリング結果と ADfeature との差異については, フィーチャーの包含関係や重複関係の扱いの差とに帰結されるとし，この結果がフィーチャーの観点に基づく分割として妥当であると判断を下した．

表 2　DMTool における性能測定結果

Table 2　Performance Measurement Results on DMTool

ソフトウェア名	クラス数	*Ka*	SArF		ACDC		Bunch*	
			MoJoFM	*K*	MoJoFM	*K*	MoJoFM	*K*
DMTool (ADpackage)	253	16	**68.6**	18	60.7	58	47.5	3-18
DMTool (ADfeature)		16	**81.4**	18	65.4	58	40.7	3-18

6. 評価

この章では，リサーチクエスチョンの検証のため，クラスタリングアルゴリズムの比較評価を行い，議論する．

比較するクラスタリングアルゴリズムは, SArF, SArF(N), Newman アルゴリズム[6] [15]（以後 Newman と略），代表的な既存研究である ACDC[4]，Bunch[2]である．Newman を選んだ理由は Newman と SArF の主な差が専念度であることから, SArF の専念度スコアが性能に貢献するかを調べるためである．SArF と SArF(N)との比較は Louvain 法の導入による得失を調べるためである．

6.1 データセット

評価用データセットのソフトウェア群はバイアスを避けるため以下に述べる客観的規準で選んだ．代表的なソフトウェアリポジトリの一つである maven リポジトリ[*2]から，以下の条件を満たすものを利用数(usages)順に取得した．

- **条件 1**: Java 言語で記述されアーキテクチャを持つ単独のソフトウェア．ツールのコレクションや，API，ラッパ，プラグイン，自身のアーキテクチャを持たない一部のライブラリやフレームワークは対象外とする．この条件の判定は人間が行う．
- **条件 2**: クラスが複数のパッケージに分散していること．1 つのパッケージに大部分のクラスが属すると，オーソリティ分割結果として適切でない[16].
- **条件 3**: 安定度の評価のため，評価対象となるバージョン数が 2 以上.

maven リポジトリはビルドツール向けのリポジトリであるため，その利用数はライブラリに偏り，アプリケーションが少なくなる．この偏りの補正のため，別の代表的なソフトウェアリポジトリである SourceForge[*3]においてダウンロード数が上位かつ前記の条件を満たすアプリケーションを maven リポジトリから取得するものに加えた．両リポジトリに対する参照と取得は 2017 年 8 月に行われた．

評価対象とするバージョンは冗長性を省くためメンテ

*2 https://mvnrepository.com/
*3 https://sourceforge.net/

ナンスバージョンが複数あるものは最新のもののみを選ぶ. 例えば ant の 1.9.x 系列は 1.9.0 から 1.9.7 までの 8 バージョンを maven から取得したが，1.9.7 のみを評価対象とした.

最終的に，maven リポジトリから 35 本（うち 10 本は SourceForge で上位のアプリケーション）のソフトウェアが選ばれ，計 304 バージョンを評価対象とした．その内訳を表 3 の先頭列に示す. 対象ソフトウェアの規模はクラス数で数十から数千まで多岐に渡り，10 倍以上に成長したソフトウェアがあるなど, 多様性のあるデータセットとなった.

6.2 測定結果

全ソフトウェア全バージョンの jar ファイルを対象に 4.3 節の手順に従って性能測定を行った．結果を表 3，図 8，図 9，図 10 に示す．SArF(N)の結果は SArF と大差が無いため，一部を除いて略した.

表 3 に各ソフトウェアの基本情報および, 各アルゴリズムのクラスタリング品質（オーソリティ準拠度とクラスタ数）の測定結果を示す．各行は各ソフトウェアの対象バージョンの測定値の平均（クラス数のみ最小値と最大値）である.「*Ka*」列はオーソリティ分割結果のクラスタ数を表す.「クラスタ数」の列の括弧の前の値は各アルゴリズムが出力した分割のクラスタ数 *K* を表し，括弧の中の数字はその *K* の MRE を表す．比較に意味がある測定値（表 3 では MoJoFM と MRE）についてはアルゴリズム間で最良のものを太字で表す（他の表も同様）. アルゴリズムごとに，全ソフトウェア全バージョンについて測定した結果を，図 8 に (a)オーソリティ準拠度，(b)クラスタ数の相対誤差，(c)安定度の箱ひげ図で示す．図 9 に *Ka* と *K* の散布図を示す．図 10 にクラス数と実行時間の散布図を示す.

6.3 議論

6.3.1 クラスタリング品質

クラスタリングの品質をオーソリティ準拠度と適切なクラスタ数の 2 点で評価する．まず，オーソリティ準拠度を見る．表 3 の MoJoFM の結果から，SArF が最良（太字）となることが多く（35 ソフトウェア中 21），ACDC と Newman が最良となるものもある．MoJoFM の最良値はソフトウェアによって 40〜70 の値をとるが，5.1 節に述べた通り MoJoFM には上限値が存在する[16]ため，必ずしもこれらの品質が低いことを意味するものではない．図 8(a)の箱ひげ図から，SArF と ACDC は，Newman や Bunch に比べ安定した値となることが判る．表 4 の上段に比較結果をまとめた．MoJoFM の平均は SArF が他より統計的有意に優れる．有意差検定には Wilcoxon の符号付順位検定（有意水準 5%）を用いた．これ以降の検定も同じ.

次に，適切なクラスタ数かどうかをみる．表 3 のクラスタ数 *K* 自身は品質に関係はなく，*K* と *Ka* との絶対相対誤差 MRE の値が重要である．MRE が小さいほど適切なクラスタ数と言える．これは SArF が最良となることが多く（35 ソフトウェア中 23），Newman と Bunch が最良となるものもある．ACDC は他と比べて MRE の値が格段と大きい．表 4 の中段に比較結果をまとめた. MRE の平均は SArF が他より統計的有意に優れる．クラスタ数について可視化したものが図 9 のバージョンごとの *Ka* と *K* の散布図である．斜線 *K*=*Ka* の付近に測定点が分布（平均 RE≒0）すればクラスタ数は適切，下に分布（平均 RE<0）すればクラスタ数過少＝クラスタサイズ過大，上に分布（平均 RE>0）すればクラスタ数過多＝クラスタサイズ過小である．ACDC が著しくクラスタ数過多になる傾向が見て取れる．これは前述のケーススタディでも確認された現象である．Newman は過少，Bunch はやや過少，SArF は適切なクラスタ数を出力する傾向があると言える．図 8(b)の箱ひげ図からも同じ結果が読み取れる.

以上から，オーソリティ準拠度とクラスタ数の 2 つの評価規準で SArF のクラスタリング品質が高いと言える．この結果は，遍在モジュール除去など人手の前処理が無い条件下のものなので，自動化を想定したものである.

6.3.2 安定度

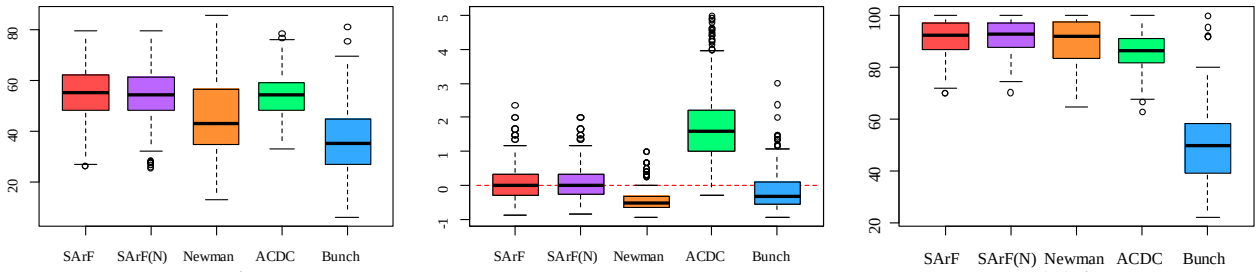
測定された安定度の分布を図 8(c)に示し，比較結果を表 4 の下段にまとめた. 安定度の平均は SArF が統計的有意に他に優れる．SArF の安定度の変動が Newman のそれに比べて小さい．これは SArF に専念度スコアを導入したことでバージョン間の重要でない変更に対して重みが減り，影響を受けにくくなった結果と考えられる.

6.3.3 実行時間

実行時間の計測環境は Xeon E5-1620v3 プロセッサ(3.50GHz)で，JVM1.8 である．クラスタリングツールは ACDC と Bunch が Java 言語で，SArF と Newman は Scala 言語で記述されており，アルゴリズム部は 1 スレッドで実行される．記述言語による実行速度の差異はほぼ無いが，ランタイム初期化時間は Scala の方が数百ミリ秒程度多い. 言語とツールの影響を除くため，実行時間には初期化時間を含めないものとした.

全ソフトウェア全バージョンの実行時間の散布図を図 10 に示す. 各アルゴリズムはクラス数 400 以下までは概ね 1 秒以内で終了し，大きな差は見られない. クラス数が 400 を越えると差が目立ち始める．クラス数に対する実行時間のオーダーをグラフの傾きとして可視化するために，図は両対数軸としている．図から明らかに，各アルゴリズムの実行時間のオーダーは Bunch＞ACDC＞Newman＞SArF(N)＞SArF と判る．特に，Bunch・ACDC と他の差異は顕著である.

表 5 は典型的な実行時間を抜粋したものである．上 4 行は前記の観測に符合する. 下 3 行は, 高速な SArF と SArF(N)の差を調べるために，1 万クラス超の規模のソフトウェアをデータセットとは別に 3 つ用意したものの測定結果である．SArF は数万クラス規模でも実行時間は数秒であるが，SArF(N)は数十秒かかり，差は明らかと言える.



(a) オーソリティ準拠度 (MoJoFM(%)) (b) クラスタ数の相対誤差 (RE) (c) 安定度 (%)

図 8　測定結果の箱ひげ図
Figure 8　Box-plot of Measurement Results

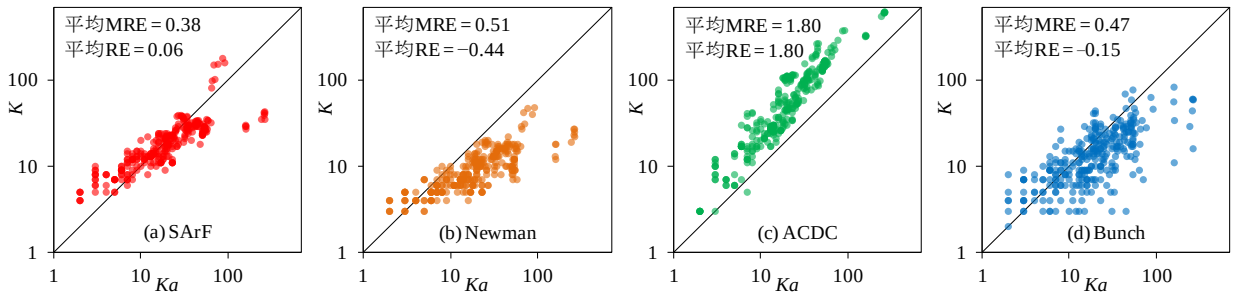


図 9　オーソリティ分割のクラスタ数 *Ka* とアルゴリズムの出力クラスタ数 *K* との比較
Figure 9　Comparisons between *Ka* of Authoritative Decomposition and *K* of Computed Decomposition

表 3　クラスタリング品質測定結果（オーソリティ準拠度とクラスタ数）
Table 3　Quality Measurement Results (Authoritativeness and Number of Clusters)

ソフトウェア名 ＋ 対象バージョン(数)	クラス数	*Ka*	オーソリティ準拠度 (MoJoFM)				クラスタ数*K*（絶対相対誤差MRE）			
			SArF	Newman	ACDC	Bunch	SArF	Newman	ACDC	Bunch
ant 1.4－1.9 (6)	158-724	18.7	**52.7**	41.4	50.2	35.0	24.5 (0.6)	9.8 **(0.4)**	49.2 (1.7)	12.3 (0.5)
aspectjweaver 1.5－1.8 (4)	281-343	7.5	**67.3**	57.4	60.7	48.7	12.0 (0.6)	7.5 **(0.1)**	37.3 (4.0)	15.3 (0.7)
avro 1.4－1.8 (5)	72-185	8.8	**62.7**	46.9	56.4	38.3	9.0 **(0.2)**	6.2 (0.3)	15.0 (0.7)	7.7 (0.4)
camel-core 2.8－2.17 (10)	879-1428	41.9	38.9	25.5	**39.9**	20.3	30.3 **(0.3)**	13.8 (0.7)	209.1 (4.0)	28.2 (0.5)
derby 10.1.1－10.12.1 (21)	1172-1453	53.2	51.1	33.6	**53.7**	28.4	25.8 **(0.5)**	10.0 (0.8)	152.9 (1.9)	27.4 **(0.5)**
easymock 2.0－3.4 (10)	55-227	3.7	71.9	**74.4**	72.3	59.5	8.3 (1.4)	4.6 **(0.4)**	11.6 (2.1)	6.4 (0.9)
elasticsearch 1.2－2.3 (10)	2876-4582	220.2	27.1	15.9	**43.1**	13.7	35.0 **(0.8)**	20.4 (0.9)	490.4 (1.2)	54.0 **(0.8)**
findbugs 1.0－3.0 (6)	431-1102	28.7	53.0	47.5	**55.2**	32.4	25.7 **(0.1)**	12.8 (0.5)	91.2 (2.1)	21.9 (0.4)
freemarker 1.5－2.3 (3)	76-444	9.0	**69.5**	57.4	63.5	50.6	12.0 (0.5)	6.3 (0.4)	24.7 (1.3)	11.3 **(0.1)**
geoserver 1.5－1.7 (3)	104-217	11.7	47.5	**52.0**	44.9	34.8	15.0 (0.3)	10.3 **(0.1)**	23.0 (0.9)	8.7 (0.4)
groovy 2.0－2.4 (5)	706-1116	50.4	36.6	27.1	**43.5**	24.5	31.8 **(0.4)**	18.0 (0.6)	114.8 (1.2)	20.8 (0.5)
gwt-servlet 1.4－2.7 (12)	316-3536	54.2	56.6	53.9	**65.9**	40.7	87.3 **(0.5)**	28.0 **(0.5)**	185.7 (2.1)	25.8 (0.6)
h2 1.2－1.4 (3)	415-490	25.7	**40.3**	24.1	38.9	24.5	18.3 **(0.3)**	6.0 (0.8)	56.3 (1.2)	15.3 (0.4)
hadoop-common 0.20－2.7 (11)	567-1092	46.8	48.7	37.0	**58.1**	30.2	29.4 **(0.4)**	16.8 (0.6)	128.0 (1.7)	20.7 (0.5)
hsqldb 1.6－2.3 (6)	52-423	10.8	**63.4**	57.2	56.8	47.9	14.7 (0.6)	6.8 (0.4)	38.8 (2.7)	12.6 **(0.3)**
httpclient 4.0－4.5 (6)	218-393	19.8	**45.1**	36.2	41.4	30.8	16.3 **(0.2)**	11.2 (0.4)	44.2 (1.2)	11.9 (0.4)
javassist 2.5－3.21 (22)	123-211	12.4	**69.3**	52.2	55.8	36.4	13.0 **(0.1)**	6.5 (0.5)	26.0 (1.1)	8.8 (0.4)
jmol 12.0－13.0 (3)	466-540	32.7	49.8	27.7	**53.5**	17.4	28.3 **(0.1)**	12.0 (0.6)	69.7 (1.1)	13.0 (0.6)
jsoup 0.2－1.10 (12)	21-54	3.0	61.4	**65.8**	56.9	54.0	5.5 (0.9)	3.7 **(0.4)**	4.7 (0.5)	5.2 (0.8)
junit 3.7－4.12 (15)	45-183	12.7	**46.9**	39.0	44.5	31.8	9.9 **(0.2)**	5.7 (0.5)	20.7 (0.6)	7.1 (0.4)
lucene-core 3.6－6.6 (25)	506-796	20.6	**59.5**	43.6	58.2	36.9	21.1 **(0.1)**	10.7 (0.5)	105.9 (4.2)	20.5 (0.4)
maven-core 2.0－3.3 (7)	54-333	13.4	51.1	51.5	**54.1**	36.9	12.9 **(0.2)**	11.3 (0.3)	29.1 (0.9)	8.9 **(0.2)**
netty 3.5－3.10 (6)	515-593	32.8	**56.9**	32.1	55.6	26.9	22.5 **(0.3)**	12.5 (0.6)	91.8 (1.8)	16.0 (0.4)
pmd 1.1－5.4 (22)	247-1067	18.4	**63.2**	54.4	61.9	43.6	15.0 **(0.2)**	7.4 (0.5)	39.3 (1.2)	13.9 (0.3)
proguard 3.4－4.4 (10)	315-500	20.2	**46.9**	30.7	40.4	28.1	10.9 (0.5)	5.3 (0.7)	33.8 (0.7)	13.4 **(0.4)**
saxon 6.5－8.7 (4)	328-705	16.3	**54.5**	33.4	48.4	27.2	14.3 **(0.2)**	6.3 (0.6)	51.3 (2.1)	16.9 (0.4)
snakeyaml 1.4－1.18 (15)	92-110	6.0	**59.8**	55.1	55.9	50.5	8.3 (0.4)	4.3 **(0.3)**	16.5 (1.8)	7.1 (0.4)
spring-web 3.1－4.3 (6)	282-390	25.5	50.8	41.8	**53.7**	26.2	27.3 **(0.1)**	19.2 (0.2)	52.0 (1.0)	11.3 (0.6)
squirrel-sql 3.0－3.5 (5)	602-755	35.2	45.0	26.8	**51.5**	27.1	27.6 **(0.2)**	15.0 (0.6)	120.4 (2.4)	20.6 (0.4)
sweethome3d 5.1－5.4. (3)	218-225	7.7	**46.6**	43.7	40.2	38.4	11.7 (0.5)	5.7 **(0.3)**	15.0 (1.0)	15.5 (1.3)
velocity 1.3－1.7 (5)	181-235	14.2	**52.8**	48.1	52.5	36.4	13.2 **(0.1)**	7.2 (0.5)	21.2 (0.5)	10.5 (0.3)
weka 3.4－3.8 (4)	676-1615	54.3	**51.4**	37.5	**51.4**	28.0	32.0 **(0.4)**	15.8 (0.7)	167.0 (2.0)	21.6 (0.7)
xalan 2.1－2.7 (6)	157-459	12.3	**72.0**	64.4	62.0	46.6	18.3 (0.5)	9.7 **(0.2)**	45.5 (2.5)	10.5 (0.3)
xercesImpl 2.0－2.11 (11)	522-709	30.2	**48.6**	38.4	48.3	30.7	37.4 **(0.2)**	13.1 (0.6)	91.9 (2.0)	14.6 (0.5)
zookeeper 3.3－3.4 (2)	153-204	6.5	**58.2**	53.9	55.0	49.3	10.0 (0.5)	7.5 **(0.1)**	35.5 (4.4)	10.9 (0.6)

実際の利用シーンにおいては，クラスタリングツールを1回実行すれば完了する用途であれば実行時間は数十秒でも問題ないが，多くの用途では有用な成果物を得るためにインタラクティブな理解プロセスを反復する場合が多く，実行時間は数秒が限度であろう．また，統合化された自動化ツールの中でクラスタリングを探索的に繰り返す用途では，実行時間が短いほどより効果が大きい．遅いアルゴリズムでも有用な用途や場面は存在する．しかし，理解対象のソフトウェアが大規模になるほど反復回数は増加する傾向にあるので，数万クラスの規模のクラスタリングが数秒で実行できる SArF の価値は高いと言える．

6.3.4 SArF の有効性

6.3.1，6.3.2，6.3.3 節の結果より，SArF はすべての指標で他のアルゴリズムより統計的有意に優れると言える．SArF と Newman の差は専念度スコアの導入に拠るため，この結果は専念度スコアの定義の妥当性の根拠となる．また，SArF が他のアルゴリズムより品質に優れることは，遍在モジュールの除去を不要にした SArF の狙いが達成されている証拠と言える．

6.3.5 SArF と SArF(N)との比較

SArF が用いるモジュラリティ最大化法は Louvain 法[18]であり，Newman アルゴリズム[15]を用いる SArF(N)[16]との得失を調べる．図 8 より SArF の品質と安定度は SArF(N)と大差ないことが判る．表 6 に検定による比較を示す．オーソリティ準拠度では SArF の方が統計的に有意に優れ，クラスタ数では差が無く，安定度では SArF(N)の方がやや(p=0.04)優れていることが判る．実行時間については 6.3.3 節より数千クラスを越える規模では差が現れる．以上から，基本的には Louvain 法を用いる SArF を使用すればよいと言える．

6.3.6 SArF とオリジナル Louvain 版 SArF との比較

3.2 節で Louvain 法を SArF に導入するに当たり，オリジナルの Louvain 法に，内部のランダム走査を固定順走査に変更する修正を施した．修正後もクラスタ数は同程度であった．その修正の効果をみるため，SArF と無修正の Louvain 法を使用した SArF(Lorig)との差を調べた結果を表 7 に示す．MoJoFM には有意差がなく，安定度は有意に SArF の方が優れた．導入した固定順走査が有効であったと言える．

6.3.7 SArF のフィーチャーに基づく分割に対する準拠度

本評価においても，SArF がフィーチャーを集めることができているかを調べる．6.3.4 節にて SArF の有効性について述べたが，各評価対象ソフトウェアのオーソリティ分割がフィーチャーや他のどの観点に基づくかは一般的には不明であるため，表 3 の結果は特定の目的に限らない「平均的」な性能の良さを表しているに過ぎない．フィーチャーの観点に基づくパッケージ階層を持つソフトウェアに対して，SArF の MoJoFM 値が特に高い場合，SArF はフィーチャーを集めていると言える．逆に，フィーチャーの観点に基づかないパッケージを持つソフトウェアに対しては，SArF は性能優位性を持たない．

表 4　品質・安定度の比較

Table 4　Quality and Stability Comparison

	SArF	Newman	ACDC	Bunch
平均MoJoFM	**54.9**	44.6	54.0	35.3
有意差※	—	有p<0.001	有p=0.03	有p<0.001
（オーソリティ分割結果の平均クラスタ数Ka = 29.9）				
平均クラスタ数K	21.6	10.3	84.5	16.7
（平均MRE）	**(0.38)**	(0.51)	(1.80)	(0.47)
平均MREの有意差※	—	有p<0.001	有p<0.001	有p<0.001
平均安定度	**91.2**	89.5	86.4	49.5
有意差※	—	有p=0.04	有p<0.001	有p<0.001

※Wilcoxon の符号付順位検定（有意水準 0.05）による，SArF との有意差．以下の表も同じ．

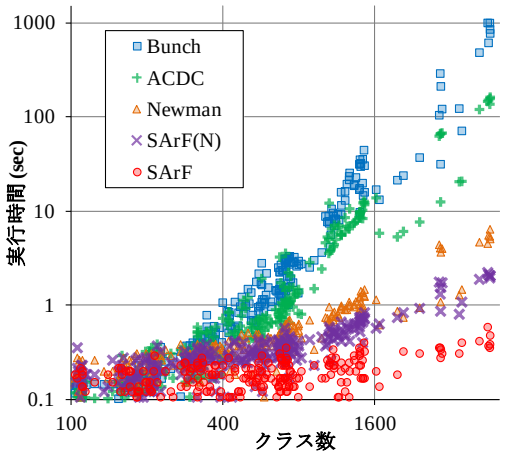


図 10　実行時間測定結果

Figure 10　Execution Time Results

表 5　実行時間測定結果（抜粋）

Table 5　Execution Time Results (Excerption)

ソフトウェア ＋バージョン	クラス数	実行時間 (sec)				
		SArF	SArF(N)	Newman	ACDC	Bunch
junit 3.7	45	0.15	0.12	0.18	0.08	0.13
maven-core 3.3.9	333	0.15	0.23	0.28	0.22	0.32
derby 10.12.1.1	1445	0.40	0.71	1.28	12.84	44.8
elasticsearch 1.4.5	4480	0.59	2.24	4.55	147.6	1007
JBoss (Wildfly) 10.1	10k	1.28	3.62			
JRE 8	19k	2.16	47.38			
Eclipse 4.7	41k	3.16	81.54			

表 6　SArF と SArF(N)の比較

Table 6　Comparison of SArF and SArF(N)

	SArF	SArF(N)	有意差
平均MoJoFM	**54.9**	54.1	有 p<0.001
平均クラスタ数	21.6	22.9	—
（平均MRE）	(0.38)	(0.37)	無
平均安定度	91.2	**91.6**	有 p=0.04

表 7　SArF とオリジナル Louvain 版 SArF の比較

Table 7　Comparison of SArF and SArF(Lorig)

	SArF	SArF(Lorig)	有意差
平均MoJoFM	54.9	54.9	無
平均安定度	**91.2**	90.4	有 p<0.001

すべての対象ソフトウェアについてパッケージ階層の観点を調べることは現実的でないため，以下の 3 ソフトウェアについて調べた．

- Javassist と xalan
  これらは，SArF の MoJoFM 値が特に高いもののうち，アーキテクチャがドキュメント化されており，パッケージ構造が基づく観点が分かるものである．両者とも，フィーチャーに基づくパッケージ階層を持つ．
- weka 3.4-3.8
  これは，SArF の MoJoFM 値が特に高いというわけではないもののうち，アーキテクチャがドキュメント化されているものである．ケーススタディ 1 で用いた weka 3.0 はフィーチャーに基づくパッケージ階層を持っており SArF が特に高い性能を出していたが，weka 3.4-3.8 ではウェブ 3 階層アーキテクチャに従ったパッケージが大量に追加され，もはやフィーチャーに基づくパッケージ階層ではなくなっており，性能優位性を失った例となっている．

上記より，確認した範囲では，SArF はフィーチャーに基づくパッケージ階層に対して準拠度の高い分割を行っていると言える．

6.4 リサーチクエスチョンの検証

4.1 節で設定したリサーチクエスチョンについて検証する．

- **RQ1**「SArF はフィーチャーを集められるか?」
  5 章の 2 つのケーススタディでは, SArF のクラスタリング結果がフィーチャーに基づくオーソリティ分割に対し高い準拠度を持つことが示された．一方，他のアルゴリズムでは準拠度が低い．また，ケーススタディ 2 では，対象ソフトウェアの開発者によって，クラスタリング結果がフィーチャーに基づくことが確認された．6 章の評価においても，6.3.7 節にて SArF のクラスタリング結果がパッケージ階層に準拠度が高い例を挙げた．これらの結果から，RQ1 は肯定的に支持されると言える．
- **RQ2**「SArF はソフトウェアクラスタリングをより自動化できたか?」
  5 章のケーススタディ 1 では, SArF が人手による遍在モジュール除去の有無に関わらず高い性能が得られ，他のアルゴリズムでは遍在モジュール除去なしでは性能が低下することが示された. 6 章の評価（6.3.1 節）では，遍在モジュール除去を行わない条件下で SArF が他より高い性能を示すことが分かった．これらから，遍在モジュール除去なしでも SArF が実用に堪え，人手の作業を減らしたと言える．よって，RQ2 が肯定的に支持されたと言える．
- **RQ3**「SArF はソフトウェアクラスタリングにおいてよい特徴を備えているか?」
  6.3.1 節から 6.3.4 節で述べたとおり, SArF はソフトウェアクラスタリングにおいて望ましい性質「クラスタリング品質」「安定性」「実行時間」において優れていることが示された．よって，RQ3 が肯定的に支持されたと言える．

7. 妥当性への脅威

内的妥当性への脅威については，まず，オーソリティ分割の問題が挙げられる．4.2 節で述べた通り，オーソリティ分割結果の入手困難さは依然解決されておらず，本論文でも既存研究と同様にパッケージ構造をオーソリティ分割結果の代替として用いている．次にオーソリティ準拠度の指標の問題が挙げられる．MoJoFM は分割過多を過大評価する欠点があり，より望ましい指標が求められる．なお，本論文で評価対象アルゴリズムがソースコードから抽出される静的な依存関係に基づくものに限っているのは，動的情報や意味的情報に基づくものとの比較は[16]で実施済みのためであり，その点に脅威は無い．

外的妥当性への脅威については，評価対象ソフトウェアが Java 言語に限られていること，MoJoFM のような階層のないフラットな分割の評価指標がどこまでの大規模ソフトウェアに通用するのか明らかでないことが挙げられる．

8. 結論

我々は静的な依存関係に基づく新しいソフトウェアクラスタリング手法 SArF を提案した．SArF は 2 つの特徴を持つ．第 1 の特徴はフィーチャーを実装するモジュールを同一クラスタ内に集めること，第 2 の特徴は人間の介入が不可欠な遍在モジュール除去作業を不要にし，ソフトウェアクラスタリングの作業をより自動化したことである．これらの特徴は専念度スコアの定義とモジュラリティ最大化法の適用により実現された．専念度の意味付けは依存するモジュールにとっての依存関係の重要度であり，専念度を用いることで遍在モジュールでないにも関わらず除去する過誤や除去し損ねる過誤を避けることができる．モジュラリティ最大化法は専念度の定義と親和性があり，専念度付きの依存関係グラフからクラスタを効率的に発見することを可能にした．

SArF の評価を行った結果を以下にまとめる．

- 公開リポジトリから客観規準（利用度順）で選んだ大規模なデータセット（35 ソフトウェア 304 バージョン）を用いて，SArF が既存の依存関係に基づくソフトウェアクラスタリングアルゴリズムに対して品質・安定度の点で優れることを示した．
- 静的な依存関係情報のみを用いてフィーチャーを集めるソフトウェアクラスタリングに成功した．
- 依存関係に基づくソフトウェアクラスタリングにおいて，遍在モジュール除去作業を不要にした．

- 実行時間の測定を行い，SArF の優位性と，数万クラスの大規模ソフトウェアのクラスタリングが数秒で完了するという高いスケーラビリティを示した．
- SArF 内部のグラフクラスタリングアルゴリズムに Louvain 法を適用する方法とその有効性を示した．

Wu ら[23]は，彼らの評価においてオーソリティ準拠度が低いことから，当時のソフトウェアクラスタリングアルゴリズムが未だ成熟していないのではと述べた．しかし，ここで示したように SArF では高い（70%以上）MoJoFM 値を得ることができ，実用に耐える性能を達成したと言える．フィーチャーを集める特徴を活かし，SArF は例えばレガシーシステムからマイクロサービスを抽出する[40]などの応用に現在利用されている．

今後の課題としては，静的な依存関係を用いるアプローチの欠点を補うために，意味的アプローチや動的アプローチとのハイブリッドなアプローチについて検討を始めている．専念度の概念は静的な依存関係に限られるものではないため，同時変更情報や開発者ネットワークなどの他のグラフで表現される情報を取り込むことは検討に値する．動的依存関係を用いた実例としては[41]がある．別の課題として，単一のモジュールが複数のフィーチャーを持つ場合に対応していないことが挙げられる．複数クラスタへの所属を許容する「ソフトな」クラスタリングアルゴリズムを組み込むことができればこの課題の解決となる．また別の課題として，ソフトウェアクラスタリングの大規模化に大きな関心が持たれる．1000 クラスを越えるソフトウェアは，フラットな分割では人間の理解が及ばず階層的な分割が必要となるであろう．そのためのアルゴリズム，可視化方法，性能評価手法などが取り組むべき課題となる．

参考文献

[1] S. Ducasse and D. Pollet: Software architecture reconstruction: a process-oriented taxonomy, *IEEE Trans. on Softw. Eng.*, vol. 35, no. 4, pp. 573-591 (2009).

[2] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner: Bunch: a clustering tool for the recovery and maintenance of software system structures, *Proc. Int'l Conf. on Softw. Maint.*, ICSM, pp. 50-59 (1999).

[3] T. Eisenbarth, R. Koschke, and D. Simon: Locating features in source code, *IEEE Trans. on Softw. Eng.*, vol. 29, no. 3, pp. 210-224 (2003).

[4] V. Tzerpos and R. C. C. Holt: ACDC: An algorithm for comprehension-driven clustering, *Proc. Working Conf. on Rev. Eng.*, WCRE, pp. 258-267 (2000).

[5] Q. Gunqun, Z. Lin, and Z. Li: Applying complex network method to software clustering, *Proc. Int'l Conf. on Computer Science and Softw. Engineering*, ICCSSE, pp. 310-316 (2008).

[6] U. Erdemir, U. Tekin, and F. Buzluca: Object oriented software clustering based on community structure, *Proc. Asia-Pacific Softw. Eng. Conf.*, APSEC, pp. 315-321 (2011).

[7] K. Praditwong, M. Harman, and X. Yao: Software Module Clustering as a Multi-Objective Search Problem, *IEEE Trans. on Softw. Eng.*, vol. 37, no. 2, pp. 264-282 (2011).

[8] G. Bavota, F. Carnevale, A. De Lucia, M. Di Penta, and R. Oliveto: Putting the developer in-the-loop: an interactive GA for software re-modularization, *Proc. Sympo. on Search Based Software Engineering*, SSBSE, pp.75-89 (2012).

[9] H. A. Müller, M. A. Orgun, S. R. Tilley, and J. S. Uhl: A reverse-engineering approach to subsystem structure identification, *J. of Softw. Maint.: Research and Practice*, vol. 5, no. 4, pp. 181-204 (1993).

[10] P. Andritsos and V. Tzerpos: Information-theoretic software clustering, *IEEE Trans. on Softw. Eng.*, vol. 31, no. 2, pp. 150-165 (2005).

[11] O. Maqbool and H. Babri: Hierarchical clustering for software architecture recovery, *IEEE Trans. on Softw. Eng.*, vol. 33, no. 11, pp. 759-780 (2007).

[12] C. Patel, A. Hamou-Lhadj, and J. Rilling: Software clustering using dynamic analysis and static dependencies, *Proc. Euro. Conf. on Softw. Maint. and Reeng.*, CSMR, pp. 27-36 (2009).

[13] Z. Wen, and V. Tzerpos: Software clustering based on omnipresent object detection, *Proc. Int'l Workshop on Program Compre.*, IWPC, pp. 269-278 (2005).

[14] 中塚剛，松下誠，井上克郎：コンポーネントランク法によるソフトウェアクラスタリング結果の理解性向上，情報処理学会論文誌，vol. 48, no. 9, pp. 3281-3285 (2007).

[15] M. Newman: Fast algorithm for detecting community structure in networks, *Physical Review E*, vol. 69, no. 6, pp. 1-5 (2004).

[16] K. Kobayashi, M. Kamimura, K. Kato, K. Yano and A. Matsuo: Feature-gathering dependency-based software clustering using dedication and modularity, *Proc. Int'l Conf. on Softw. Maint.*, ICSM, pp. 462-471 (2012).

[17] A. Clauset, M. Newman, and C. Moore: Finding community structure in very large networks, *Physical Review E*, vol. 70, no. 6 (2004).

[18] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre: Fast unfolding of communities in large networks, *J. Stat. Mech. Theory Exp.*, vol. 2008, no. 10, p. P10008 (2008).

[19] N. Anquetil and T.C. Lethbridge: Recovering software architecture from the names of source files, *J. of Softw. Maint.: Research and Practice*, vol. 11, pp. 201-221 (1999).

[20] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico: Using the Kleinberg algorithm and vector space model for software system clustering, *Proc. Int'l Conf. on Program Compre.*, ICPC, pp. 180-189 (2010).

[21] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello: Weighing lexical information for software clustering in the context of architecture recovery, *Empir. Softw. Eng.*, vol. 21, no. 1, pp.72-103 (2016).

[22] T. Richner and S. Ducasse: Using dynamic information for the iterative recovery of collaborations and roles, *Proc. Int'l Conf. on Softw. Maint.*, ICSM, pp. 34-43 (2002).

[23] J. Wu, A. E. Hassan, and R. C. Holt: Comparison of clustering algorithms in the context of software evolution, *Proc. Int'l Conf. on Softw. Maint.*, ICSM, pp. 525-535 (2005).

[24] R. A. Bittencourt and D. D. S. Guerrero: Comparison of graph clustering algorithms for recovering software architecture module views, *Proc. Euro. Conf. on Softw. Maint. and Reeng.*, CSMR, pp. 251-254 (2009).

[25] M. Girvan and M. E. J. Newman: Community structure in social and biological networks., *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 12, pp. 7821-7826 (2002).

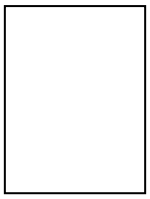
[26] V. Tzerpos and R. C. Holt: MoJo: A distance metric for software clusterings, *Proc. Working Conf. on Reverse Eng.*, WCRE, pp. 187-193 (1999).

[27] Z. Wen, and V. Tzerpos: An effectiveness measure for software clustering algorithms, *Proc. Int'l Workshop on Prog. Compre.*, IWPC, pp. 194-203 (2004).

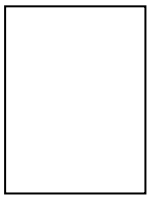
[28] M. Shtern and V. Tzerpos: Lossless comparison of nested software

decompositions, *Proc. Working Conf. on Rev. Eng.*, WCRE, pp. 249-258 (2007).
[29] M. Shtern and V. Tzerpos: On the comparability of software clustering algorithms, *Proc. Int'l Conf. on Program Compre.*, ICPC, pp. 64-67 (2010).
[30] M. Shtern and V. Tzerpos: Refining clustering evaluation using structure indicators, *Proc. Int'l Conf. on Softw. Maint.*, ICSM, pp. 297-305 (2009).
[31] T. Lutellier, D. Chollak, J. Garcia, et al.: Measuring the Impact of Code Dependencies on Software Architecture Recovery Techniques, *IEEE Trans. Softw. Eng.*, doi:10.1109/TSE.2017.2671865 (2017).
[32] M. Newman and M. Girvan: Finding and evaluating community structure in networks, *Physical Review E*, vol. 69, no. 2 (2004).
[33] E. A. Leicht and M. E. J. Newman: Community structure in directed networks, *Physical Review Letters*, vol. 100, no. 11, p. 118703, (2008).
[34] A. Arenas, J. Duch, A. Fernández, A., and S. Gómez: Size reduction of complex networks preserving modularity, New Journal of Physics, vol.176, no.9, pp.1-15 (2007).
[35] U. Brandes, D. Delling, M. Gaertler, et al.: On modularity clustering, *IEEE Trans. on Knowledge and Data Engineering*, vol. 20, no. 2, pp. 172-188 (2008).
[36] A. Lancichinetti and S. Fortunato: Community detection algorithms: A comparative analysis, *Physical Review E*, vol. 80, no. 5 (2009).
[37] I. Stavropoulou, M. Grigoriou, and K. Kontogiannis: Case study on which relations to use for clustering-based software architecture recovery, *Empir. Softw. Eng.*, vol. 22, no. 4, pp. 1717-1762 (2017).
[38] H. Witten and E. Frank: *Data Mining Practical machine learning tools and techniques*, Morgan Kaufmann (2005).
[39] S. Ducasse, T. Girba, and A. Kuhn: Distribution map, *Proc. Int'l Conf. on Softw. Maint.*, ICSM, pp. 203-212 (2006).
[40] M. Kamimura, K. Yano, T. Hatano and A. Matsuo: "Extracting Candidates of Microservices from Monolithic Application Code," Proc. Asia-Pacific Software Engineering Conference, (APSEC), pp. 571-580 (2018).
[41] K. Yano and A. Matsuo: "Data Access Visualization for Legacy Application Maintenance," Proc. Int'l Conf. on Software Analysis, Evolution and Reengineering, (SANER), pp. 546-550 (2017).

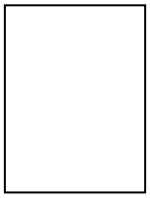
著者紹介



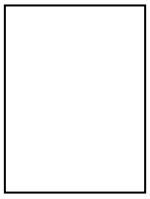
小林健一（正会員）
平 6 東京大学大学院工学系研究科修士課程修了．同年富士通研究所入社．令 3 大阪大学大学院情報科学研究科後期博士課程修了．現在まで富士通株式会社勤務．博士（情報科学）．AI 品質，ソフトウェア保守，プログラム理解，HPC の研究開発に従事．



松尾昭彦（正会員）
昭 62 東京理科大学理学部物理学科卒．同年（株）富士通研究所入社．業務アプリケーションの保守や再構築に関する研究開発に従事．令 4 退職．平 26-27 情報処理学会理事．



松下誠（正会員）
平 10 大阪大学大学院基礎工学研究科博士後期課程修了．同年同大学基礎工学部情報工学科助手．平 14 大阪大学大学院情報科学研究科コンピュータサイエンス専攻助手．平 17 同専攻助教授．平 19 同専攻准教授．博士（工学）．リポジトリマイニング，プログラム解析の研究に従事．



井上克郎（正会員）
昭 59 大阪大学大学院基礎工学研究科博士後期課程了（工学博士）．同年大阪大学基礎工学部情報工学科助手．昭 59-61 ハワイ大学マノア校コンピュータサイエンス学科助教授．平 3 大阪大学基礎工学部助教授．平 7 同学部教授．平 14 同大学大学院情報科学研究科教授．令 4 より南山大学理工学部教授．ソフトウェア工学，特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事．