依存関係グラフを用いた
ソフトウェア保守の効率化に関する研究

提出先　　大阪大学大学院情報科学研究科

提出年月　2021 年 7 月

小 林　健 一

内容梗概

現代の社会基盤を支えているソフトウェアシステムには，十年を超える長寿命のものが珍しくなく，その多くが大規模かつ複雑化している．そしてこれからも，新たな商機の出現や社会環境の変化に対応するために，常に進化が求められ続けている．そこで求められているソフトウェア保守の活動には欠陥修正や機能追加，マイグレーションなどが挙げられるが，これらの活動は常に，熟練プログラマの引退やドキュメントの損失・陳腐化など，知識損失の危機に曝されている．このような危機的な背景の下で，ソフトウェア保守を滞りなく，効率的に実施するための技術が必要となる．

本研究は，ソフトウェア保守において知識損失の影響を受けにくいソースコードから抽出される依存関係情報を用いることで，多くの局面で適用可能な，ソフトウェア保守の効率化のための新たな技術を提案するものである．特に，ソフトウェア保守の重要な活動である欠陥修正と機能追加・マイグレーションの２つに着目して研究を行った．

まず，欠陥修正を支援する目的では，以下の２つの研究を行った．

- 依存関係グラフに基づく影響波及量の計測法と，それを用いた欠陥予測手法
- 工数予測における対数変換の効果分析と精度向上のための補正方法

これらにより，欠陥修正箇所の特定や，工数の優先付けが容易になり，限られた人的資源を効率的に保守に役立てることが可能となった．

次に，機能追加・マイグレーションを支援する目的では，失われたソフトウェアのアーキテクチャ情報を復元して理解する研究として，以下の２つの研究を行った．

- 依存関係グラフを用いたソフトウェアアーキテクチャの復元手法
- 依存関係グラフを用いたソフトウェアアーキテクチャの可視化手法

これらにより，従来人手では多大な時間のかかっていた，ソフトウェアアーキテクチャという高い抽象度の情報の復元が可能となり，またそれを人間が理解しやすいメタファーを用いて可視化することで，保守期間中に知識が失われた，または現状と乖離したソフトウェアアーキテクチャの姿を理解することができ，機能追加やマイグレーションの大きな支援となる．

これらの手法により，大規模ソフトウェアの保守という社会基盤を維持するうえで欠かせない活動に対し，一定の効率化を果たすことができたと考える．

論文一覧

主要論文

[1] 小林健一，松尾昭彦，井上克郎，早瀬康裕，上村学，吉野利明：大規模ソフトウェア保守のための影響波及量尺度インパクトスケール，情報処理学会論文誌，Vol. 54，No. 2，pp. 870-882 (2013)．情報処理学会論文賞．

[2] 門田暁人，小林健一：線形重回帰モデルを用いたソフトウェア開発工数予測における対数変換の効果，コンピュータソフトウェア誌，Vol.27，No.4，pp. 234-239 (2010).

[3] 小林健一，松尾昭彦，松下誠，井上克郎：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，（投稿中）.

国際会議

[4] K. Kobayashi, A. Matsuo, K. Inoue, Y. Hayase, M. Kamimura, and T. Yoshino: ImpactScale: Quantifying change impact to predict faults in large software systems, IEEE International Conference on Software Maintenance (ICSM), pp. 43-52 (2011).

[5] K. Kobayashi, M. Kamimura, K. Kato, K. Yano, and A. Matsuo: Feature-gathering dependency-based software clustering using dedication and modularity, IEEE International Conference on Software Maintenance (ICSM), pp.462-471 (2012).

[6] K. Kobayashi, M. Kamimura, K. Yano, K. Kato, and A. Matsuo: SArF Map: Visualizing Software Architecture from Feature and Layer Viewpoints, IEEE International Conference on Program Comprehension (ICPC), pp.43-52 (2013).

関連論文

[7] 小林健一，松尾昭彦，井上克郎，早瀬康裕，上村学，吉野利明 : インパクトスケールを用いた大規模ソフトウェア保守の障害予測，情報処理学会ソフトウェアエンジニアリングシンポジウム 2010 (2010).

[8] 小林健一 : メタファーを用いた高抽象度ソフトウェア可視化実践の予備報告，情報処理学会ソフトウェア工学研究会ウィンターワークショップ 2014・イン・大洗 (2014).

[9] 小林健一，吉野利明，井上克郎，早瀬康裕，松尾昭彦，上村学 : 保守の影響波及範囲に基づいたレガシーシステムの障害予測，電子情報通信学会技術研究報告，SS2005-68，Vol. 105，No. 491，pp. 31-36 (2005).

[10] 早瀬康裕，松下誠，楠本真二，井上克郎，小林健一，吉野利明 : 影響波及解析を利用した保守作業の労力見積りに用いるメトリックスの提案，電子情報通信学会論文誌 D，Vol.J90-D，No.10，pp. 2736-2745 (2007).

[11] 松尾昭彦，小林健一，多門宏晋 : アプリケーション保守向けメトリクス「インパクトスケール」による品質評価技術，電気学会 電子・情報・システム部門大会 (2009).

謝辞

本研究の推進にあたり，ご指導を賜わり，適切なるご助言を戴きました，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井上克郎教授に，心より深謝申し上げます．井上教授には 2004 年より共同研究の機会を戴き，以来長年にわたり，ソフトウェア工学分野において多面的なご指導を賜るだけに留まらず，研究者としての心構えや活動など様々な面において多くのものを学ばせて戴きました．私が今日，一端の研究者足りえるのは井上教授のご指導の賜物と考えます．

本研究の推進にあたり，ご指導とご助言を戴きました，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松下誠准教授に心より御礼申し上げます．長年の共同研究を通じて，松下准教授の多岐に渡る幅広い知識と鋭い着眼点は，私の研究を進める際の大きな助けとなりました．

本学位論文の副査としてご助言を戴きました，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増澤利光教授と楠本真二教授に感謝申し上げます．増澤教授には，学位論文中の論旨の不明瞭な点を丁寧にご指摘戴き，論文完成への大きな助けとなりました．楠本教授には，井上教授とともに共同研究を当初から進めさせて戴き，その中で長年のご指導を戴きました．ソフトウェア工学の実証的研究について，技法から考え方まで，本研究を進めるための根幹となる多くのものを学ばせて戴きました．

岡山大学門田暁人教授には，3 章の工数予測に関する研究を共に進めさせて戴きました．ソフトウェア工学における様々な予測問題について深く議論する機会を持てたことは研究者としての人生における喜びとなりました．篤く御礼申し上げます．

また，井上教授と松下准教授が率いられます井上研究室の方々にも，本研究の推進にあたり，多大なるお世話を戴きました．石尾隆助教（現，奈良先端科学技術大学院大学准教授）には，ソフトウェア解析について様々な助言を戴くとともに，ソフトウェア可視化のワークショップを共同運営させて戴きました．早瀬康裕氏（現，筑波大学助教）には，共同研究の初期から共に同じ影響波及の問題に取り組み，苦労を分かち合い，2 章の研究に至りました．また，肥後芳樹氏（現，大阪大学准教授），吉田則裕氏（現，名古屋大学准教授）らを初めとする学生諸氏との研究議論は新たな着想が生まれる楽しい場でありました．そのような場に共にあることができたことに感謝を述べたいと思います．そして，研究室の諸事を引き受け，研究の負担を和らげて頂いた軽部瑞穂氏にお礼を述べさせて戴きます．

訪問研究員として訪れたオーストラリアの NICTA (National ICT Australia)では，多くの有益な助言と，海外の一流の研究者に触れる機会を得ました．そこでお世話になり，深く議論頂いた Dr. Liam O'Brien，Dr. Liming Zhu（現，University of New South Wales 教授），Dr. Jacky Keung（現，City University of Hong Kong 准教授）に感謝致します．特に，Dr. O'Brien にはソフトウェアアーキテクチャ復元の研究についてご教授いただき，それが 4 章 5 章の研究の端緒となりました．

同時期に NICTA の訪問研究員であった亀井靖高氏（現，九州大学准教授）には，折に触れソフトウェア保守の様々なトピックについて議論する機会を戴き，本研究では，2 章中の工数考慮モデルについて参考にさせて戴きました．ここで感謝を述べさせて戴きます．

東洋大学野中誠教授には，NICTA 訪問へのアドバイスを戴き，2 章の研究成果を現場にて実証し新たな知見を引き出して戴けました．ここで感謝を述べさせて戴きます．

また，本研究の実現にあたり，株式会社富士通研究所ならびに富士通株式会社の皆様には様々なご支援とご協力を賜りましたことを感謝致します．その中でも直接研究を支援し，深く関わって戴いた，富士通執行役員常務 原裕貴氏，元富士通フィールド・イノベーション本部フィールドイノベーター 吉野利明氏，富士通アプリケーションシェアードサービス統括部シニアディレクター 松尾昭彦氏，同マネージャー 上村学氏，同 前田芳晴氏，同 矢野啓介氏，富士通アドバンスドテクノロジー推進統括部 加藤光幾氏，富士通データ＆セキュリティ研究所 佐々木裕介氏にこの場でお礼を述べさせて戴きます．特に松尾氏には，本研究の全期間を通して，適切な助言と研究遂行のための様々な支援を戴きました．篤く感謝を述べさせて戴きます．また，学位取得活動を支援頂いた富士通フェロー 岡本青史氏，富士通人工知能研究所所長 穴井宏和氏，同プロジェクトディレクター 栗原英俊氏に感謝を述べさせて戴きます．

最後に，常日頃，社会人としての勤務を続けながらの学位取得活動で苦労を掛けること多々ある中，私を支え続けてくれた妻，小林麗に感謝します．そして，学位取得活動の期間中にこの世に生を受け，我々夫婦に大いなる喜びをもたらしてくれ，笑顔と元気で我々の人生を彩ってくれた最愛の息子，小林恒陽に永遠の感謝を贈ります．

目次

1. 緒言 9
1.1 ソフトウェア保守とその課題 9
1.2 本研究の目的 10
1.3 本論文の構成 11
2. 依存関係グラフを用いた欠陥予測 12
2.1 はじめに 12
2.2 インパクトスケール 14
2.3 評価 18
2.4 考察 27
2.5 関連研究 31
2.6 まとめ 32
3. 工数予測の精度向上のため対数変換と補正方法 33
3.1 はじめに 33
3.2 対数変換を伴う線形重回帰モデル 34
3.3 プロジェクトデータの特徴からの考察 37
3.4 ケーススタディ 38
3.5 まとめ 41
4. 依存関係グラフを用いたソフトウェアアーキテクチャの復元 42
4.1 はじめに 42
4.2 関連研究 43
4.3 SArF ソフトウェアクラスタリングアルゴリズム 45
4.4 実験計画 48
4.5 ケーススタディ 50
4-6 評価 53
4.7 妥当性への脅威 59
4.8 まとめ 59
5. 依存関係グラフを用いたソフトウェアアーキテクチャの可視化 61
5.1 ソフトウェア保守におけるプログラム理解と可視化 61
5.2 関連研究 62
5.3 ソフトウェア可視化手法 SArF Map 63

5.4 リサーチクエスチョン 71
5.5 ケーススタディ 72
5.6 妥当性への脅威 80
5.7 まとめと今後の課題 80
6. 結言 83
参考文献 85

1. 緒言

現代の社会基盤を支えているソフトウェアシステムには，十年を超える長寿命のものが多く，大規模化かつ複雑化しており，それらの価値を減耗することなく，トラブルなく提供を続けることはますます困難になってきている．そしてこれからも，社会環境の変化や新たな商機の出現に対応するために，ソフトウェアシステムは常に進化が求められ続けていく．本研究は，このようにソフトウェアシステムを維持し改善していくソフトウェア保守の各活動について，その支援を行い，効率化を図る技術を提案するものである．

1.1 ソフトウェア保守とその課題

ソフトウェア保守とは，IEEE/ISO/IEC 14764-2006 の定義によれば，ソフトウェア提供後に行われる，欠陥の修正や予防のため，性能改善や他の特性の改善のため，または変化する環境へ適応させるためのソフトウェアの修正であり，そして，そのための活動の全体とある．例えば，欠陥の修正のためには，障害発生の検出，障害レポートの報告や収集，原因の調査，修正方法の特定，修正の実施，正しく修正が行われたかを確認するテスト，退行が起きていないかのリグレッションテストなど，多くの活動がそこには含まれる．これらの効率化手法や，支援ツールの開発，分析などが求められている．また，修正は典型的には人間が行うため，人的要因として知識とコストの問題も考慮に入れる必要がある．

一方，変化する環境への適応のためには，修正に関する活動に加え，対象システムの新たな要求・要件の定義，適応のための変更規模の決定，すなわち軽微な改善に留めるのか，機能を追加するのか，マイグレーションなど大幅な刷新を行うのか，など高レベルの判断が問われる場合がある．この場合，サービス対象のドメイン知識やソフトウェアアーキテクチャなどの高抽象度知識が必要となる．

知識の問題に焦点をあてると，ソフトウェアシステムに関する知識は，ソースコードのように，実装されたものが実行され，様々なテストや検証などによって正しさが継続的に担保されつづけるものもあれば，要件定義書や設計書などのドキュメントのように，実装に比べ抽象度が高く，リリース後には正しさや実装との一貫性を維持する動機が少ないものもある．また，開発者の経験などドキュメント化されていないものもある．リリース後，時間が経つにつれ，ドキュメントと実装の乖離は進む傾向があり，経験などの属人的知識は開発者の異動や離職などで失われる機会が多い．このように，ソフトウェア保守においては知識の損失が継続的に発生する問題がある．

コストの問題について焦点をあてると，一般に，ソフトウェア保守では年間予算などコストに制約があることが多い．すべての欠陥を障害発生前に発見し修正することは現実的には不可能であり，限られた予算や工数を，実施すべき修正活動の候補に対して優先付けして割り当てることが必要になる．しかしこの優先付けの判断は実際には難しい．障害のもたらす経済的損失の規模や，修正にかかる工数などを事前に見積ることは

困難であるし，前述の知識損失により十分な判断材料を揃えることが現実的には難しい．

1.2 本研究の目的

本研究では，前節で挙げたソフトウェア保守における課題に取り組み，解決するための技術を提案する．そのような技術の有用性を考える上では，その適用可能性も問われる．特に，継続的に知識損失が起きるソフトウェア保守の環境下で，十全の効果を発揮する技術が望ましい．よって，提案技術に対する要請として，本研究では，ソフトウェア保守で不可避の課題となる知識損失に対して頑健であることを求めた．そこで，知識損失の影響を受けにくいソースコードから抽出可能な情報，具体的には，ソースコードに内在する呼び出し関係やアクセス関係などの依存関係を活用する．これら依存関係の集合はグラフ構造をなすため，これを依存関係グラフと呼ぶ．この依存関係グラフはソフトウェア保守のほとんどの局面で抽出が可能であり，これに基づいて構築された技術は高い適用可能性を持つ．

この依存関係グラフを用いて，ソフトウェア保守の活動のうち，以下について新たな技術を提案し，それにより保守の効率化を図るというのが本研究の目的である．

(1) 修正の影響波及量の測定と，それを用いた欠陥予測
(2) 工数予測
(3) ソフトウェアアーキテクチャの知識の復元
(4) ソフトウェアの高レベル知識の理解

これらの研究項目の重要性について述べる．(1)と(2)は欠陥修正に関わるもので，まず(1)の修正の影響波及量とは，各々のソースコードやクラスについて，修正の際にどれほどの影響が他に及ぶかの知識である．これが大きければ修正の難易度や工数は増える上に，そもそも障害を引き起こす原因となる可能性が高くなる．これはドキュメントに残されていない隠れた知識であり，これを説明変数として欠陥予測や工数予測を行うことで，従来にない精度での予測が可能になる．

次に(2)について述べる．前節で述べたとおり，限られたコストや工数などのリソースを効率よく活用するには，優先順位付けが重要であり，精度の高い工数予測により正確な見積りが得られればそれが可能になる．(1)で得た影響波及量は精度向上に寄与するが，予測モデル自身も適切な措置を施せば精度向上の機会がある．

(3)と(4)は機能追加やマイグレーションに関わるもので，それらの方針を決定する上で必要となる高レベル抽象度の情報を(3)で抽出し，(4)で意思決定がその情報を理解するのを支援する．(3)で抽出される情報は，ドキュメントに残されていないか，残されていたとしても現在の実装とは乖離した昔の情報だけという，保守において失われやすい典型的な知識である．この情報の有無は，意思決定の妥当性や見積もりの精度に大きく関わることになる．また，この情報は，保守開発者が個人の知識として保有しドキュメント化されていな

いものであるので，それを明らかにし，他の保守開発者に共有することで，欠陥修正の諸活動にも役だつ.

1.3 本論文の構成

本論文では，前節に述べた各研究項目(1)から(4)についての研究結果を，それぞれ 2 章から 5 章までで述べ，最後に 6 章で結言を述べる．以下，2 章から 5 章までの概要を述べる．

まず，2 章は，依存関係グラフに基づく影響波及量の計測法と，それを用いた欠陥予測手法についての研究である．ソフトウェア保守においては変更時に他に及ぼす影響の大きいモジュールほど扱いが難しい．この章では欠陥予測の精度向上を目的として，依存関係グラフに基づいて，変更の影響波及量を定量化するメトリクス「インパクトスケール」を提案し，二つの大規模企業システムを対象として欠陥予測を行い，その有効性を評価する．

次に，3 章は，工数予測における対数変換の効果分析と精度向上のための補正方法である．ソフトウェアに関する工数予測では，線形重回帰モデルは基本的な予測モデルとして多く用いられてきた．この章では，対数変換を行った線形回帰モデルは，指数曲線モデルと等価であり，ソフトウェア開発データの特徴を現すのに適していることを示す．ただし，対数変換を行ってから線形回帰を行うと，逆変換する際に過小予測するバイアスが発生してしまうため，その補正方法について述べる．

次いで，4 章は，依存関係グラフを用いたソフトウェアアーキテクチャの復元手法である．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ために役立つ．この章では，依存関係グラフ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．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，オープンソースソフトウェアを対象として SArF を評価し，その有効性を評価する．

そして，5 章は，依存関係グラフを用いたソフトウェアアーキテクチャの可視化手法である．この章では，ソフトウェアシステムのアーキテクチャの理解を容易にするために，都市メタファーを用いてフィーチャーとレイヤーの二つの観点からソフトウェアアーキテクチャを可視化する SArF Map を提案する．SArF Map は，4 章で述べた SArF を用いてフィーチャーを抽出し，可視化する．フィーチャーはソフトウェアの高レベルな抽象化単位であるため，生成されたビューは再利用などの高レベルな意思決定に直接利用できるほか，開発者と非開発者であるステークホルダーの間のコミュニケーションにも役立つ．ケーススタディを通じて，SArF Map によってソフトウェアのアーキテクチャの理解と品質評価が容易にできることを示す.

2. 依存関係グラフを用いた欠陥予測

2.1 はじめに

ソフトウェアの欠陥予測はレビュー，テスト，品質及びリスク管理を支援する手段として大きく成果を上げている[2-1][2-2]．しかし，リリース後や保守フェーズにおける欠陥予測はリリース前に比べ困難となる．欠陥予測には欠陥と関係の強いメトリクスが説明変数として必要であるが，リリース前欠陥と相関が高いメトリクス群とリリース後障と相関が高いメトリクス群は異なり[2-3][2-4]，リリース前に有効なメトリクスはリリース後や保守では多くの場合その有効性を失うためである．例えば Fenton ら[2-3]は，リリース前には McCabe の循環的複雑度[2-5]と欠陥との間に相関があったが，同じソフトウェアのリリース後には相関がほぼ無かったという事例を報告している．

保守では，ソースコードから採取されるプロダクトメトリクスだけでは実用的な性能の欠陥予測が難しいため，ソフトウェアプロセス・欠陥履歴・変更履歴などから採取されるプロセスメトリクスを併用して予測性能を上げる試みがなされている[2-6]．例えば，過去に変更が行われた箇所は他より欠陥が起こりやすいという観測に基づく研究が行われている[2-4][2-6][2-7].

しかしながら，現実の保守では経年に伴うドキュメントの陳腐化や人材の流出，ベンダの変更などでプロセスメトリクスが採取できない場合が多く，プロダクトメトリクスのみで性能の高い欠陥予測ができることは実用上の価値が高い．そのためにはソースコードから欠陥要因と関係の強い情報を採取する必要がある．長く保守が続くソフトウェアでも除去が難しい欠陥要因のひとつにソフトウェア内の依存関係がある[2-8][2-9]．ソフトウェア内のモジュールが変更された時，依存するモジュールにも変更が必要になる可能性がある．その変更はまた新たな変更を招き，すべての必要な変更が完了するまでソフトウェア内を波及していく．この波及効果[2-10]の見逃しは欠陥発生の典型的な一因である．Hassan ら[2-9]は論理的結合関係(logical coupling)[2-8]が変更波及の認識に利用できると報告している．論理的結合関係とはソフトウェア内の同時変更関係であり，同時変更により暗黙の依存関係が観測されたとみなせる．論理的結合関係の強度が欠陥と関係があることがよく知られている[2-6].

Gall らはソフトウェアの依存関係の分類を行い，ソースコードに記述された直接の依存関係を構文的依存関係とし，プロセスから採取される論理的結合関係などを論理的依存関係とした[2-8]．構文的依存関係はプロダクトメトリクスにより，論理的依存関係はプロセスメトリクスにより計測される．Cataldo ら[2-11]は様々な依存関係と欠陥との相関を比較し，構文的依存関係と欠陥との間の相関は有意ではないまたは弱く，論理的依存関係と欠陥は有意であり最も強かったと報告している．この報告は，既存のプロダクトメトリクスだけでは保守の欠陥予測には不十分であるという一事例である．

Zimmerman と Nagappan[2-12]はソーシャルネットワーク解析(SNA)をソフトウェアのバイナリモジュール間の依存関係グラフに対して行い，SNA 分野からネットワーク尺度を導入し予測モデルに追加することで欠陥予測の性能が向上すると報告した．ネットワー

ク尺度はプロダクトメトリクスではあるが，前述の Cataldo らによる構文的依存関係の分類の中には含まれない．

本研究では，Zimmerman らによって用いられたネットワーク尺度と，Cataldo らの論理的依存関係の分類が欠陥潜在性の共通要素を示していると仮定した．そして，ソースコードの影響波及解析[2-13]を行えば，論理的結合関係によって曝されるような暗黙の依存関係を抽出できると仮定した．影響波及解析はソフトウェアのある部分が変更された場合にその影響を受ける範囲を同定する技術である．Cataldo らの実験では，対象モジュールに関わる論理的結合関係の数がそのモジュールの欠陥回数と最も相関が強かった．よって，変更の影響範囲の大きさの推定値も同様に欠陥回数と相関が強いと期待し，以下の仮説を立てた．

仮説 1: 影響波及量を定量化したメトリクスは大規模ソフトウェアにおいて欠陥予測の性能を向上させることができる．

しかし実際には，影響波及範囲を正確に求めることは難しい．静的解析による影響波及解析[2-14]には偽陽性の範囲を過剰に導出する短所があり，精度を上げるためには莫大な計算量を必要とする．動的解析による影響波及解析[2-15]では偽陽性を抑え影響範囲を絞りこむことが可能であるが，利用頻度が稀な箇所は実行履歴に残りにくく，影響範囲の見落としを招くという短所を持つ．また，実稼働システムでは，運用上の理由から動的解析を実施できない場合も多い．対象モジュールへの変更内容が事前に判っている場合には影響範囲を求める実践的な技法も提案されているが[2-16]，本研究の目的のためには影響波及範囲は変更を実施する前に知る必要がある．結論として，偽陽性を抑えつつ影響波及範囲を求めることは困難と言える．

ここで，仮説 1 の焦点は影響波及範囲を正確に求めることではなく，より緩和された問題である「影響波及の大きさの推定」であることに注目する．本研究では，影響が確率的に波及し，かつ二項間に定義された関係に依存して波及を制御する制約を持つ波及モデルを定義し，そのモデル上で影響波及範囲の広さを推定することとした．この推定値を，影響波及量を表すメトリクス「インパクトスケール」と名付けた[2-17]．このインパクトスケールの妥当性は，以下の仮説の上に立脚する．

仮説 2: 確率的かつ関係依存的な波及を持つ波及モデルは，正確だが計算量やデータ収集コストが大きい解析の代わりとなるのに十分な予測性能を持つ．

本章の構成を以下に示す．2.2 節で影響波及量を表す新しいメトリクス「インパクトスケール」を定義する．2.3 節では既存の予測モデルにインパクトスケールを追加することで予測性能が上昇するかを評価する．2.4 節ではインパクトスケールの効果と定義の妥当性について考察し，2.5 節では関連研究について述べ，最後に 2.6 節でまとめる．

2.2 インパクトスケール

本節では，新しいメトリクス「インパクトスケール」(ImpactScale, IS)を定義し，その基盤となる影響波及モデルについて述べる．

2.2.1 依存グラフと波及グラフ

まず本研究で用いる影響波及モデルについて述べる．ソフトウェアの構成要素であるコードエンティティ（メソッドや関数など）やデータエンティティ（データベースのテーブルやグローバル変数など）をノードとし，ノード間の依存関係を辺（本研究では，辺はすべて有向辺）とするグラフ構造で表したものを依存グラフと呼ぶ．依存関係には，コール依存やデータ依存など様々な種類があり，この種類を「関係タイプ」と呼ぶ．依存グラフはノード間の辺が多重であってもよい．依存グラフ G_D は G_D=<V, E>で定義される．V はノードの集合，E は辺の集合である．辺 e∈E は e = <s, t, rel>で定義される．s∈V は始点ノード，t∈V は終点ノード，rel∈Rel は関係タイプである．Rel は関係タイプの値域集合である．

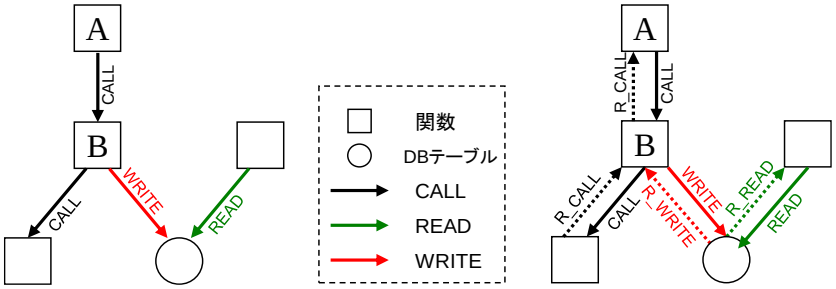


図 2-1　依存グラフ G_D（図左）と波及グラフ G_P（図右）の例

図 2-1 左に依存グラフの一例を示す．図の依存グラフには，CALL，READ，WRITE の 3 種の関係タイプがある．図の関数 A が関数 B をコールするとき，関数 A から関数 B への CALL 関係があると呼ぶ．関数 A と関数 B は状態を共有したり実装を分担したりする可能性があり，A から B への波及も B から A への波及も可能性を排除できない．離れたエンティティ間の論理的依存関係を見逃さないためには，波及が双方向に起こりうるという保守的な戦略を採る必要がある．

波及グラフは依存グラフに依存グラフの辺を反転させた辺を追加したものである．依存グラフ G_D=<V, E>に対応する波及グラフは G_P=<V, E_P>と定義される．E_P は E_P = E ∪ {reverse(e) | e∈E} で定義される．reverse(e)は辺 e = <s, t, rel>を反転した辺 e' = <t, s, R_rel>を意味する．関係タイプ R_rel は rel を反転した関係タイプを意味する．影響波及量の計算はこの波及グラフの上で行われる．図 2-1 右は図 2-1 左の依存グラフ G_D に対応する波及グラフ G_P である．

2.2.2 確率的な波及

Haney[2-10]は，モジュールの変更が他のモジュールに確率的に伝わるという解析モデルを提案し，同様の仮定をおいた研究が行われてきた[2-18][2-19]．本研究でも同様の仮定を置く．例えば，図 2-2 の波及グラフにおいて，ノード v_0 からノード v_1 への辺 e_1 があるとき，v_0 から v_1 への影響は波及率 r_1 の確率で伝わるとする．同様に，v_1 から v_2 への影響は波及率 r_2 の確率で，v_0 から v_2 への影響は r_1 r_2 の確率で伝わるとする．

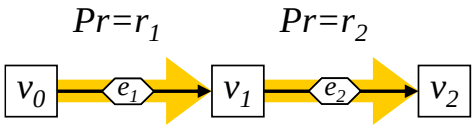


図 2-2　確率的な波及の例

2.2.3 関係依存的な(relation-sensitive)波及

前述の通り波及の追跡戦略を保守的としたため，影響範囲が過剰に見積もられる恐れがある．これを抑制するための制約を波及追跡戦略に課す．例えばコールグラフ解析では，解の精度を上げるために，経路探索の打ち切り（以降，カットと呼ぶ）に過去のコール経路や状態を参照するコンテクスト依存(context-sensitive)解析がしばしば利用される[2-14]．コンテクスト依存解析は計算量が大きくそのままでは利用できないが，本研究ではそこから解決のための着想を得た．

計算量の爆発を抑えるために，現実的に収集可能かつ追加の計算量が最小となるコンテクストである「前の辺の関係タイプ」を用いる．具体的には，あるノードから次の辺への波及のカットの是非を，前の辺の関係タイプと次の辺の関係タイプの組を定義域とする写像 Cut: Rel×Rel ↦ {True, False}で決定する．例えば，図 2-3 に示す波及グラフと写像 Cut において，ノード v の前の辺 e_1 の関係タイプが rel_{prev1}，次の辺 e_3 の関係タイプが rel_{next3} の時，Cut(rel_{prev1}, rel_{next3})は True なので波及はカットされる．この関係タイプによって制御された波及を「関係依存的な(relation-sensitive) 波及」と呼ぶ．

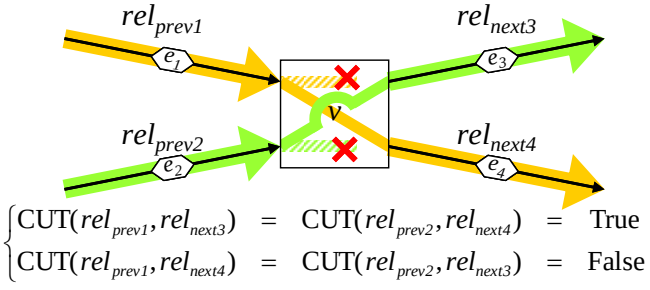


図 2-3　関係依存的な波及の例

2.2.4 インパクトスケールの定義

本節では，インパクトスケールの定義を示す．波及グラフ G_P=<V, E_P>上で，ノード s からノード t へのあるパス p を，その経由する辺 e_i を用いて p=(e_1, e_2, ... e_n)と表す．p は辺の経由順の順序列であり，e_1 の始点が s，e_n の終点が t である．このパスは厳密にはグラフ理論の用語のトレイル(trail)であり，同じノードを複数回通過しても構わない．辺 e_i の波及率 r_i は次式で与えられる．

$$r_i = R_P \cdot m(e_i)$$

R_P は基本波及率であり本章では 0.5 としている．m(e_i)は辺 e_i に付随する波及補正値であり，(0,1]の範囲または 1/R_P の値を取るが，通常は 1 である．パス p を介す影響波及量 $Q_{path}(p)$は次式で与えられる．

$$Q_{path}(p) = \frac{1}{R_P}\prod_{i=1}^{n} r_i = {R_P}^{n-1}\prod_{i=1}^{n} m(e_i)$$

s から t への到達可能なすべてのパスの集合を Ps,t とする．Ps,t は波及グラフを探索して求められる．探索中にあるノードに到達した時の次の辺への波及の是非は，写像 Cut で決定される．すなわち，Ps,t は波及グラフと写像 Cut により定まる．なお，始点から次の辺へは必ず波及するとする．s から t への影響波及量 Q(s, t)は次式で与えられる．

$$Q(s,t) = \begin{cases} \max_{p \in P_{s,t}} Q_{path}(p) & : \ P_{s,t} \neq \varphi \\ 0 & : \ P_{s,t} = \varphi \end{cases}$$

Q(s, t)の計算は本質的には最短経路問題の求解と等価である．s のインパクトスケールを IS(s)と表すと，IS(s)は次式で与えられる（V¥s は V から s を除いた部分集合の意）．

$$IS(s) = \sum_{t \in V \backslash s} Q(s,t)$$

以上からなる四つ組<IS, R_P, Rel, Cut>が，インパクトスケールの定義である．

2.2.5 カットルール

関係タイプの値域集合 Rel と写像 Cut は抽出する依存関係と目的に応じて自由に設定でき，本章では，Rel={CALL, READ, WRITE, R_CALL, R_READ, R_WRITE}とする．CALL はコール，READ はリードアクセス，WRITE はライトアクセス（リードアクセスも含意する）を

意味し，R_CALL, R_READ, R_WRITE はそれぞれ反転された関係タイプである．写像 Cut は次式を用いる．

$$\mathrm{Cut}(p,n)=\begin{cases}\mathrm{True} & : \quad p=\mathrm{CALL}\wedge n=\mathrm{R\_CALL} \quad \cdots \quad (\mathrm{Rule1})\\ \mathrm{True} & : \quad p=\mathrm{R\_CALL}\wedge n=\mathrm{CALL} \quad \cdots \quad (\mathrm{Rule2})\\ \mathrm{True} & : \quad p=\mathrm{READ} \quad \cdots \quad (\mathrm{Rule3})\\ \mathrm{False} & : \quad \mathrm{otherwise}\end{cases}$$

写像 Cut の上の表記の条件部をカットルールと呼ぶ.

前記のカットルールは発見的であり，その妥当性は 2.4 節で議論するが，ここではその意図を述べる．CALL から CALL への波及はトップダウン設計に沿って上位モジュールの変更が下位モジュールに波及することを想定し，R_CALL から R_CALL への波及はボトムアップ設計に沿って下位モジュールの変更が上位モジュールに波及することを想定している．Rule1 は CALL から R_CALL への波及をカットし，Rule2 は R_CALL から CALL への波及をカットするものであるが，これらは，一方の設計に沿っていながら，他方の設計へ切り替えることは起きにくいという推測から設けた．Rule3 は，データフロー解析からの類推から設けた.

2.2.6 計算量

インパクトスケールの計算の大部分は経路探索に占められる．波及グラフは多重辺を持つグラフであるため，通常のグラフアルゴリズムは適用できない．Whaley ら[2-20]はコンテクスト依存解析のために，ノード複製によりコンテクスト付きの複雑な経路探索問題をコンテクスト無しの単純な経路探索問題に帰結する方法を示した．この方法を用いると，インパクトスケールの経路探索も多重辺の無い単純なグラフ上の経路探索問題に帰結でき，通常のアルゴリズムが適用できる．ダイクストラの最短経路アルゴリズムを波及グラフ G_P=<V, E_P>に適用した場合，R を関係タイプの種類数とすれば，計算量は $O(R|E_P| + R|V|(\log(R|V|))$である．これは対象システムが大規模であっても実用的な時間で計算可能である.

2.2.7 計算例

図 2-4 左はノード C のインパクトスケールの計算例である．パス(C→A)，(C→D)，(C→X)，(C→X, X→F)，(C→X, X→F, F→E)，(C→X, X→F, F→H)が探索されている．辺 C→A の関係タイプは R_CALL で，辺 A→B の関係タイプは CALL であるため，パス(C→A, A→B)は Rule2 によって辺 A→B の箇所でカットされる．同様に，パス(C→X, X→F, F→H, H→G)は Rule1 によって辺 H→G の箇所でカットされる．C のインパクトスケールは Q(C,A), Q(C,D), Q(C,X), Q(C,F), Q(C,E), Q(C,H)の合計であり，4.0 である.

図 2-4 右はノード J のインパクトスケールの計算例であり，「関係依存的な波及」の特徴的なケースが現れている．(J→K, K→L)というパスは Rule1 により辺 K→L の箇所でカ

ットされるが，別のパス(J→K, K→Y, Y→K, K→L)というパスはカットされない．結果として，J からの影響は L まで波及する．J のインパクトスケールは Q(J,K), Q(J,Y), Q(J,L)の合計であり，1.625 である．データノード Y は J から L への媒介として働くため，Y が無ければ K までしか波及せず J のインパクトスケールは 1.0 となってしまう．

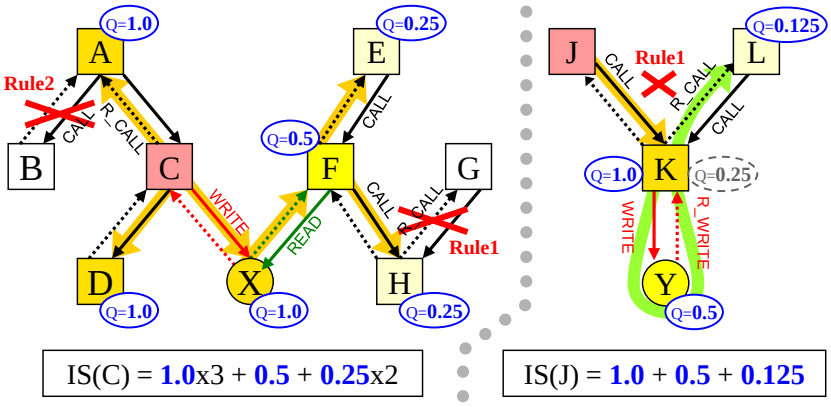


図 2-4　インパクトスケールの計算例

2.2.8 用途

インパクトスケールの用途として，例えば以下のものが想定される．

- 限られた予算と期間の下での欠陥予測を用いた品質改善．次節以降でこれについて詳しく述べる．
- バグ修正作業の早期見積もり．修正対象となるソースコードからの影響範囲の大きさは，コードレビューや回帰テストなど修正作業の工数に直接影響するため．
- ソフトウェアシステム全体の早期の品質監査．
- ソフトウェアのモジュラリティを維持するためのコード修正作業の監視．インパクトスケールの目立った上昇は，モジュラリティ違反の兆候である．

2.3 評価

仮説 1 と仮説 2 の検証と，インパクトスケールの有効性を評価するため，以下のリサーチクエスチョンを設定する．

RQ1: インパクトスケールを既存のプロダクトメトリクスに加えることは欠陥予測性能を向上させられるか？

RQ2: インパクトスケールを既存のプロダクトメトリクスとネットワーク尺度を併せたものに加えることは欠陥予測性能を向上させられるか？

続く小節の構成は以下の通りである．

- 2.3.1 節～2.3.3 節では実験の設定について説明する．
- RQ1 については，欠陥予測性能の評価を，2.3.4 節にて二項判別を用いて行い，2.3.5 節にて工数考慮モデル[2-21]を用いて行う．
- RQ2 については，2.3.6 節にて工数考慮モデルと階層モデル分析を用いて行う．

2.3.1 評価対象ソフトウェア

本研究では評価対象として，二つの企業から大規模な勘定系ソフトウェアシステムのデータを入手した．これらを選んだ規準は，長期間保守されており，適用領域における標準的な規模を持ち，そして，ソースコード解析の精度の影響をこの評価において最小化するために解析の容易な言語で記述されていることである．二つのシステムから収集したデータセットを DS1，DS2 と称し，そのプロファイルを表 2-1 に示す．両システムは COBOL 言語で記述されており，それぞれについて近年の 40 ヶ月分の欠陥レポートを収集した．

表 2-1　評価対象ソフトウェアのプロファイル

名前	モジュール数	合計 LOC	欠陥数	欠陥モジュール数
DS1	5.8k	1.6M	269	215
DS2	7.6k	3.7M	250	208

本評価では，1 モジュールは，COBOL 言語の「プログラム」であり，ソースファイル 1 本である．「プログラム」はコール命令の対象となる単位で，他言語の関数に相当する．採集したメトリクスを表 2-2 に示す．「セクション」は COBOL 言語特有の概念で，関数内部のサブブロックに相当する．

表 2-2　採集したメトリクス

メトリクス	説明
LOC	コメント・空行を除いたコード行数
WMC	モジュール全体の McCabe の循環的複雑度
MaxVG	セクション毎の McCabe の循環的複雑度の最大値
Sections	セクション数（モジュール内のブロック数に相当）
Calls	コール命令の数
Fan-in	モジュールを呼出すモジュールの数
Fan-out	モジュールによって呼出されるモジュールの数
IS	インパクトスケール

2.3.2 インパクトスケールの測定

インパクトスケールの測定は以下の手順で行われる．まず，測定対象ソフトウェアから静的解析にてコール命令(CALL)と，データベースやファイルへのアクセス操作(READ, WRITE)を抽出する．次に，依存グラフと波及グラフを構築する．そして，その依存グラフの全てのノードについて，インパクトスケールを前述の定義に従い計算する．ここでは m(*e*)はすべて 1.0 とした．対象システムにおける計算時間は数十秒であった（使用した CPU は Core2Duo 2.5GHz）．図 2-5 に測定されたインパクトスケールの分布と統計量を示す．図中，ほとんどのモジュールが小さいインパクトスケールの値を持ち，ごく一部が大きな値を持つ．著者の経験では，この傾向はほとんどの大規模ソフトウェアにおいて共通する．

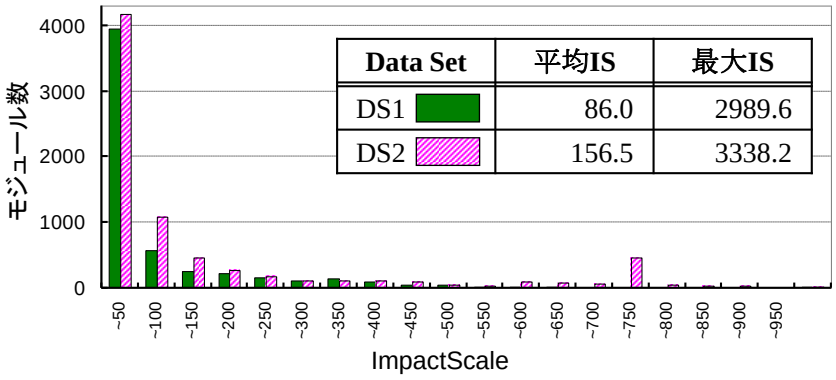


図 2-5　測定されたインパクトスケール

2.3.3 反復検証

結果の妥当性を保証するため，各データセットの各評価につき 100 回のランダムサブサンプリングバリデーションを行った．100 回の反復のそれぞれにおいて，対象データセットの全モジュール中の 2/3 を訓練セットとしてランダムに選び，残りの 1/3 のモジュールをテストセットとして用いて，予測性能の測定を行う．比較対象群の差異の有意性検定には，標本サイズ 100 に対する Wilcoxon の符号順位検定を用いた．

2.3.4 二項判別 (RQ1)

本節では，インパクトスケールを既存のプロダクトメトリクス群に追加することで，予測性能向上が得られるかを評価するために，二項判別を行う．

2.3.4.1 ロジスティック回帰

分類器としてロジスティック回帰モデルを用いる．ロジスティック回帰モデルは次式で表される．

$$y = \frac{\exp(b_1 x_1 + b_2 x_2 + ... + b_0)}{1 + \exp(b_1 x_1 + b_2 x_2 + ... + b_0)}$$

この時，y は目的変数，各 x_i は説明変数，各 b_i は偏回帰係数である．式 $b_1x_1+b_2x_2+ ... +b_0$ は線形予測子と呼ばれる．偏回帰係数を推定するためには最尤推定法を用いる．目的変数 y ($0<y<1$)は欠陥潜在確率と解釈される．

2.3.4.2 モデル選択

各反復において，各訓練セットに対する最良の予測モデルを選択するため変数選択法を述べる．表 2-2 に挙げたメトリクスが予測モデルの説明変数の候補であり，メトリクスの対数値（値域に 0 を含む場合は 1 を加算後の対数値）も代替候補とした．すなわち，各メトリクスにつき，値・対数値・非選択の三択である．偏った分布のデータに対数変換を施す措置は標準的に用いられる技法である．モデルの選択規準には AIC（赤池情報量規準）[2-22]を用いた．AIC はモデルの当てはまり度合い（尤度）に説明変数の数によるペナルティを課したもので，AIC が小さいほどモデルが良いことを示す規準である．モデルの説明変数としてインパクトスケール抜きの 7 つ，またはインパクトスケールを含めた 8 つメトリクスのすべての組み合わせ（3^7 通り，または 3^8 通り）から，AIC 最小となる組み合わせを選んだ．多重共線性を避けるために，説明変数のいずれかの VIF (Variance Inflation Factor) が 10 以上となったモデルは除外した[2-23].

2.3.4.3 モデル学習とテスト

各反復において，表 2-2 に挙げたインパクトケール以外の既存メトリクスを用いて最良モデルを学習したものが「モデル MET」である．同様に既存メトリクスとインパクトスケールを併せて最良モデルを学習したものが「モデル MET+IS」である．学習とテストでは，欠陥数が 1 以上であれば欠陥有り，欠陥数が 0 ならば欠陥無しとして扱った．

次に，テストセットに対して予測を行った．テストセットの各モジュールに対し，目的変数 y が所定の閾値以上ならばそのモジュールを欠陥モジュールと分類した．DS1 と DS2 の両方とも欠陥率が低い（3.7%と 2.7%）ため，閾値は 0.1 とした．

2.3.4.4 性能尺度

予測性能の評価のため，各バリデーションにおいて適合率(Precision)と再現率(Recall)と F1 値を用いる．TP を欠陥有りと観測され欠陥有りと予測されたモジュール数，TN を欠陥無しと観測され欠陥無しと予測されたモジュール数，FP を欠陥無しと観測され欠陥有りと予測されたモジュール数，FN を欠陥有りと観測され欠陥無しと予測されたモジュー

ル数とすると，適合率は TP/(TP+FP)と定義され，1.0 に近い適合率は欠陥有りと誤って判定されるモジュールがほとんど無いことを意味する．再現率は TP/(TP+FN)と定義され，1.0 に近い再現率はほとんどの欠陥が検出されることを意味する．F1 値は適合率と再現率の二値の要約尺度で，二値の調和平均（適合率の逆数と再現率の逆数の平均の逆数）で定義される．

2.3.4.5 結果

表 2-3 に 100 回の反復結果の各性能尺度の平均値と標準偏差を示す．「モデル MET」と「モデル MET+IS」の列はそれぞれのモデルでの性能尺度，「IS による向上」は二つのモデルの差，つまりインパクトスケールを説明変数に追加したことによる向上である．表から，インパクトスケールの追加は両データセットで全ての性能尺度を有意に向上させていることが判る．この結果は RQ1 を肯定的に支持する．

表 2-3　二項判別における欠陥予測性能の向上

データセット	性能	モデル MET		モデル MET+IS		IS による向上†
	尺度	平均	標準偏差	平均	標準偏差	
DS1	適合率	0.148	(0.029)	0.168	(0.031)	+0.020
	再現率	0.315	(0.051)	0.392	(0.048)	+0.077
	F1	0.200	(0.033)	0.234	(0.034)	+0.034
DS2	適合率	0.139	(0.030)	0.162	(0.033)	+0.023
	再現率	0.253	(0.042)	0.334	(0.057)	+0.081
	F1	0.177	(0.029)	0.216	(0.034)	+0.039

†全ての向上は Wilcoxon の符号順位検定で有意 ($P<0.001$)

2.3.5 工数考慮モデルによる評価 (RQ1)

2.3.5.1 工数考慮モデルと性能尺度

近年，モジュールのテストや監査にかかる工数を欠陥予測の性能評価でも考慮すべきと指摘されている[2-21][2-24][2-25][2-26]．欠陥と規模には相関がある[2-3]ため，欠陥有りと分類されるモジュールは規模が大きい傾向がある．例えば Arisholm らはモジュールのテスト工数はそのモジュールの規模に大まかに比例すると報告している[2-24]．実際の保守では，予算とスケジュールは多くの場合要求が厳しく，欠陥予測の工数効率性は実務者にとっての重要な関心となってきている．

上記の議論を踏まえ，本研究では予測性能評価に Mende ら[2-21]によって提案された工数考慮モデルを用いた．工数考慮モデルでは，相対リスク $R_{dd}(x)$ がモジュールのテストや監査の優先付けに用いられる．$R_{dd}(x)$ は $\#errors(x)$ をモジュール x の欠陥数，$E(x)$ をモジュール x に必要な工数としたとき，$\#errors(x) / E(x)$ と定義される．ここでは既存研究[2-21][2-26]と同様に $E(x)$ として LOC を用いる．この場合，$R_{dd}(x)$ は欠陥密度を意味

する．工数考慮モデルでは $R_{dd}(x)$（即ち，欠陥密度）を予測し，モジュールを欠陥密度の降順でテストまたは監査を行う．予測性能を調べるためには，図 2-6 に例示される工数ベース累積リフトチャートが用いられる．図の実曲線が総合的な予測性能を示している．この曲線はコスト効果曲線(cost-effectiveness curve)[2-24]，または工数対検出率曲線(effort-vs-PD curve)[2-25]と呼ばれる．X 軸は相対累積工数を表し，Y 軸は費やした工数での欠陥検出率を表している．曲線の立ち上がりが急ならば，予測はより工数効率が高いことを意味している．曲線が対角線を下回るならば，予測がほとんどランダムで意味の無いことを表す．「完全予測曲線」と題された点線曲線は，完全な予測が行われた場合を示しており，性能の上限値を表している．

図 2-6 は本研究で使用する二つの性能尺度についても説明している．「AUC」(Area Under the effort-vs-PD Curve)は工数対検出率曲線の下部領域の面積を意味する[2-25]. AUC は全領域での平均性能を表しており，AUC が 1.0 以下の上限値に近ければ，予測性能が高いことを意味する．一方，AUC が 0.5 前後かそれ以下ならば，その予測に意義は無い．「ddr」(defect detection rate)は欠陥検出率である[2-21]．本研究では「ddr x」は全工数の x%において検出した欠陥率を意味する．保守では典型的に工数が逼迫しており，この尺度は実践者の例えば「10%の行数をレビューするとどれだけの欠陥を検出できるか？」等の問いに直接答えることができる．高い ddr x は高い予測性能を意味する．既存研究には x として 20%を用いるものがあるが，大規模システムにおける実践的見地からは 20%は大き過ぎる故に，本研究では ddr_{10} の方を ddr_{20} よりも重視する．もし ddr_{10} が 0.1 より小さければ，予測による利得が無いことを意味する．

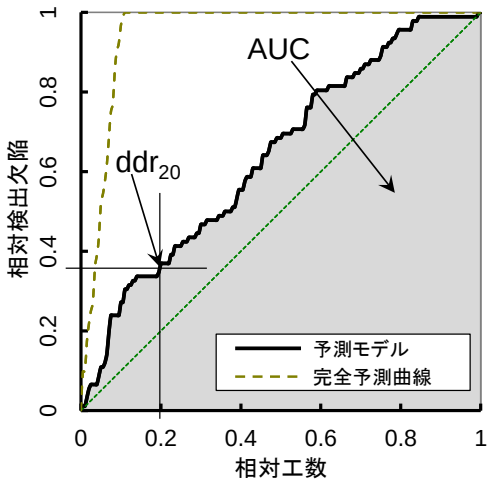


図 2-6　工数ベース累積リフトチャートと AUC，ddr の例

2.3.5.2 ポアソン回帰分析を用いた欠陥密度予測

欠陥密度予測のためには連続値か計数値が出力となる予測モデルが必要となる．本研究では，ポアソン回帰モデル[2-27]を使用する．ポアソン回帰モデルは確率事象のカウン

ト予測に一般的に用いられるモデルであり，欠陥発生をポアソン過程に従う確率事象とみなすことにより従来から欠陥予測の分野で用いられている[2-6][2-28]．ポアソン回帰モデルの式は一般的には次式で表される．

$$y = \exp(b_1 x_1 + b_2 x_2 + \ldots + b_0)$$

偏回帰係数 b_i を推定するためには最尤推定法を用いる．目的変数 y $(0<y)$ は欠陥回数の期待値と解釈される．目的変数が発生回数なので発生密度は直ちに導出可能である．密度を予測するためには係数を 1.0 に固定したオフセット項 log(LOC) を線形予測子に以下の通り加えて学習されればよい．

$$y = \exp(b_1 x_1 + b_2 x_2 + \ldots + b_0 + \log(\mathrm{LOC}))$$

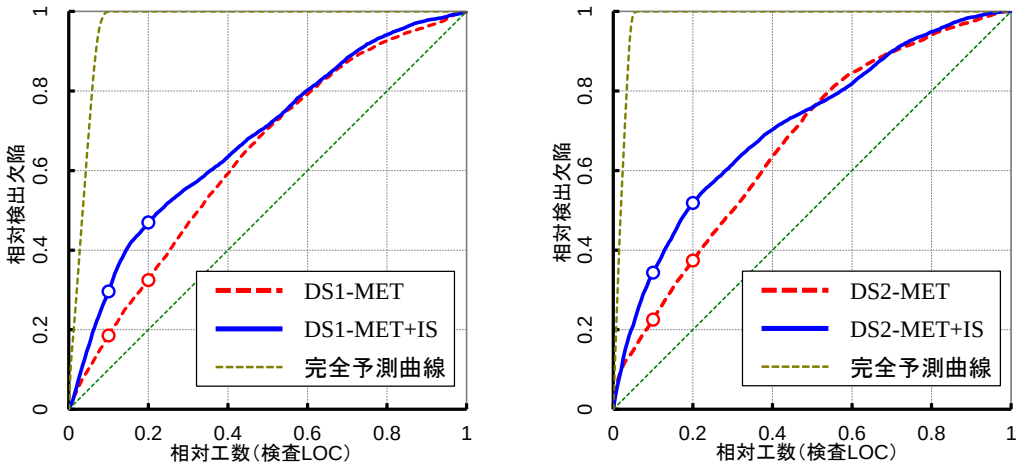
オフセット項の存在に関わらず，log(LOC) を説明変数に加えることが可能である．

2.3.5.3 モデル選択，モデル学習とテスト

最良モデルの選択および，モデルの学習とテストは 2.3.4 節と同様に行われる．差異は，欠陥数を学習に直接用いることのみである．

2.3.5.4 結果

図 2-7 と表 2-4 に 100 回の反復結果を示す．図中，赤い破線の各曲線がインパクトスケール無しのモデルの性能を表す（DS1-MET／DS2-MET）．青い実線の各曲線がインパクトスケール有りのモデルの性能を表す（DS-MET+IS／DS2-MET+IS）．曲線上の白い中抜き点は表 2-4 にある ddr_{10} と ddr_{20} の測定点である．DS1 と DS2 の両方で，図ではモデル MET+IS の曲線はモデル MET の曲線をほぼすべての範囲で上回り，表では全ての性能尺度で MET+IS が上回る．特に，ddr_{10} の向上は顕著である．この結果は，全検査工数の 10%の工数において，検出できる欠陥数に 1.5 倍の向上が得られることを意味する．この結果もまた，RQ1 を肯定的に支持する．



(a) DS1の工数ベース累積リフトチャート　(b) DS2の工数ベース累積リフトチャート

図 2-7　インパクトスケール有り／無しモデルの工数考慮モデル比較

表 2-4　欠陥密度と工数考慮モデルにおける欠陥予測性能の向上

性能尺度	モデル DS1-MET		モデル DS1-MET+IS		IS による向上†
	平均	標準偏差	平均	標準偏差	
AUC	0.635	(0.027)	0.680	(0.027)	+0.045
ddr_{10}	0.186	(0.042)	0.296	(0.051)	1.60 倍
ddr_{20}	0.325	(0.043)	0.470	(0.055)	1.45 倍

AUC の上限値は 0.977.　†全ての向上は Wilcoxon の符号順位検定で有意 (P<0.001)

性能尺度	モデル DS2-MET		モデル DS2-MET+IS		IS による向上‡
	平均	標準偏差	平均	標準偏差	
AUC	0.669	(0.025)	0.714	(0.025)	+0.045
ddr_{10}	0.225	(0.047)	0.343	(0.046)	1.53 倍
ddr_{20}	0.374	(0.053)	0.518	(0.051)	1.39 倍

AUC の上限値は 0.984.　‡全ての向上は Wilcoxon の符号順位検定で有意 (P<0.001)

2.3.6 ネットワーク尺度との比較 (RQ2)

Zimmermann と Nagappan らは，ソーシャルネットワーク解析(SNA)を依存グラフに行い，欠陥予測に利用した[2-12]．彼らの研究と追試[2-29][2-30][2-31]は，ネットワーク尺度の有効性を示した．インパクトスケールは依存グラフ上で測定されるため，その有効性を確認するために RQ2 を調べる必要がある．

RQ2（再掲）： インパクトスケールを既存のプロダクトメトリクスとネットワーク尺度を併せたものに加えることは欠陥予測性能を向上させられるか？

2.3.6.1 SNA のネットワーク尺度

ネットワーク尺度はネットワークの様々なトポロジー的特徴量である．Zimmermann らは 58 のネットワーク尺度を UCINet ツール[2-32]を用いて収集し，欠陥予測を行った．使用されたネットワーク尺度の完全なリストと説明は[2-12]と[2-33]を参照のこと．本章では既存研究[2-12][2-29][2-30][2-31]と同じく UCINet を用い，インパクトスケールの測定に用いた DS1 の依存グラフに対してネットワーク尺度を収集した．DS2 の依存グラフは UCINet が扱える規模を超えているため，本節の評価では DS1 のみを対象とした．

2.3.6.2 主成分回帰分析

本節でもポアソン回帰分析を用いて欠陥密度予測を行い，工数考慮モデルを用いて評価を行う．この場合，説明変数が多く（60 以上）多重共線性は避けられないため，既存研究と同様に主成分分析(PCA)[2-34]を用いる．PCA は主成分を発見するための教師無し学習である．すべての主成分は直交するため，主成分を説明変数として用いる回帰分析では多重共線性の問題は発生しない．この組み合わせは主成分回帰分析と呼ばれる．

2.3.6.3 モデル選択，モデル学習とテスト

以下の 4 つのモデルを用意した．

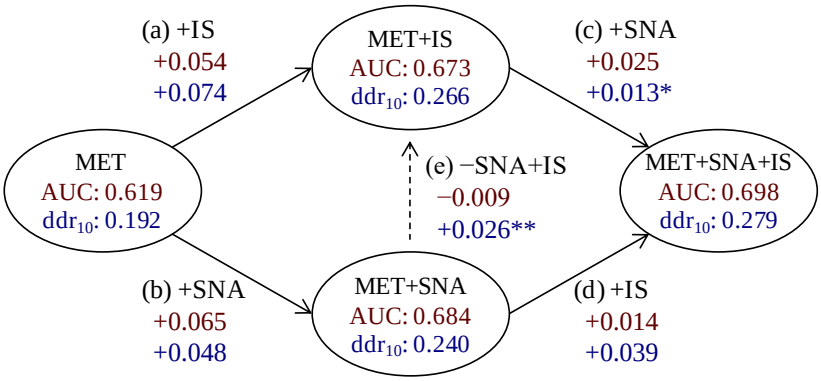
- MET　既存メトリクスからなる最良モデル
- MET+IS　既存メトリクスとインパクトスケールからなる最良モデル
- MET+SNA　既存メトリクスとネットワーク尺度からなる最良モデル
- MET+SNA+IS　既存メトリクス，ネットワーク尺度，インパクトスケールからなる最良モデル

各反復において，4 つのモデルごとに，全メトリクスと全尺度を対数変換し PCA を用いて主成分に変換する．そのうち，累積寄与率 99%までの主成分をポアソン回帰分析の説明変数として使用する．使用された主成分数の平均値は MET で 6.0，MET+IS で 7.0，MET+SNA で 27.8，MET+SNA+IS で 28.8 であった．テスト手順は 2.3.5 節と同じである．

2.3.6.4 結果

図 2-8 は，結果を階層モデル比較で示したものである．4 つの楕円は 4 つの予測モデルを表し，それぞれの性能尺度 AUC と ddr_{10} が記されている．実線矢印は説明変数の追加によるモデルの拡張を表しており，付された数字は上が AUC の向上，下が ddr_{10} の向上を表している．例えば，矢印(a)はインパクトスケールをモデル MET に追加し，予測性能が AUC で 0.054 向上，ddr_{10} で 0.074 向上したことを表している．これは 2.3.4 節，2.3.5 節の結果に整合する．矢印(b)はネットワーク尺度の効果を表し，既存の SNA を用いた研究の結果に整合する．

RQ2 を調べるため，矢印(d)と矢印(e)に注目する．矢印(d)はモデル MET+SNA にインパクトスケールを追加することが有意であることを表している．これはインパクトスケールがネットワーク尺度とは異なる欠陥要因説明要素を含んでいることを意味している．矢印(e)はインパクトスケールとネットワーク尺度との比較であり，ddr_{10} が増加していることからインパクトスケールが工数に限りがあるときにネットワーク尺度よりも高い検出率を持つと言える．一方，AUC が減少していることから工数に際限がない想定では予測性能が若干低くなると言える．いずれにせよ，インパクトスケール単独がもたらす予測性能の向上はネットワーク尺度の集合全体による向上に匹敵し，これは予測モデルの解釈容易性という点で大きな利点となる．以上から，RQ2 は肯定的に支持される．ネットワーク尺度については関連研究の節にてさらに述べる．



すべての向上と低下はWilcoxonの符号順位検定で有意（*: P<0.05, **: P<0.01, 無印: P<0.001）

図 2-8　インパクトスケールとネットワーク尺度の階層モデル比較

2.4 考察

本節では，インパクトスケールが確かに欠陥予測に寄与しているかどうか，2.2 節で与えたインパクトスケールの定義が妥当かについて考察する．

2.4.1 欠陥予測への寄与

まず，インパクトスケールと他のメトリクスとの関係を調べる．DS1 についての，表 2-2 のメトリクス間のスピアマンの順位相関係数を表 2-5 に示す．絶対値が 0.5 以上のものを太字で示している．インパクトスケールは他のメトリクスとの相関係数の絶対値が最大でも高々0.38 であり明らかに低い．この結果は DS2 でも同様であった．ネットワーク尺度に関しては，インパクトスケールとの相関係数は高々0.60 であり，図 2-7 に示した向上に寄与するほど十分に低い．これはインパクトスケールが他のメトリクスとは独立，つまり他のメトリクスの示さない欠陥要素を示していると言える．

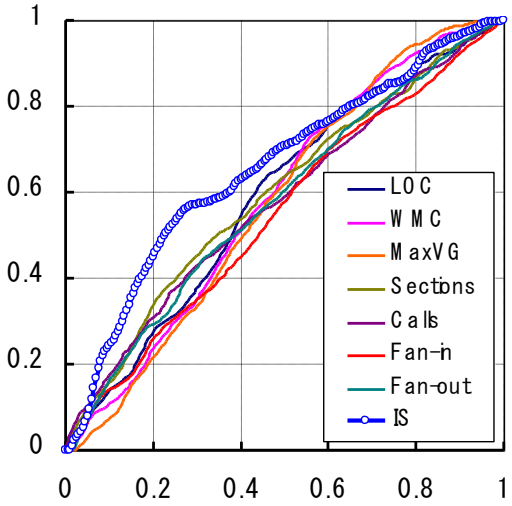
表 2-5　メトリクス間のスピアマンの順位相関係数

	WMC	MaxVG	Section	Calls	Fan-in	Fan-out	IS
LOC	0.90	0.71	0.80	0.69	-0.26	0.66	0.18
WMC	-	0.88	0.59	0.51	-0.13	0.48	0.11
MaxVG	-	-	0.35	0.34	-0.05	0.32	0.04
Section	-	-	-	0.78	-0.33	0.79	0.31
Calls	-	-	-	-	-0.32	0.94	0.33
Fan-in	-	-	-	-	-	-0.32	0.17
Fan-out	-	-	-	-	-	-	0.38

次に，DS1 で，各メトリクスによる 1 変数ポアソン回帰にて欠陥密度予測を行った結果（100 回平均）が図 2-9 の工数ベース累積リフトチャートと図中の表である．インパクトスケール以外のメトリクスは AUC が 0.5 をやや上回る程度，ddr_{10} は 0.1 前後であり予測が無判別に近いのに対し，インパクトスケールは単独で高い予測性能を発揮している．よって，インパクトスケールは DS1 において欠陥予測に強く寄与していると言える．DS2 も同様であった．

RQ1 と RQ2 に関する結果と本節から，仮説 1 は本研究において妥当と言える．

仮説 1（再掲）： 影響波及量を定量化したメトリクスは大規模ソフトウェアにおいて欠陥予測の性能を向上させることができる．



Metrics	AUC	ddr_{10}
LOC	0.591	0.137
WMC	0.586	0.107
MaxVG	0.578	0.073
Sections	0.596	0.156
Calls	0.579	0.173
Fan-in	0.549	0.139
Fan-out	0.580	0.162
IS	**0.654**	**0.242**

図 2-9　1 変数欠陥密度予測の結果

2.4.2 インパクトスケールの定義の妥当性

ここでは，インパクトスケールの定義が妥当かどうか検証する．インパクトスケールの定義を変化させ，それに対し予測性能がどう変化するかを，ddr_{10} を基準に調べる．

まず，依存グラフ上でグラフ距離が離れた遠隔ノードを考慮することの意義を調べる．インパクトスケールは経路探索により計算される．そのため，経路探索の距離に上限を設けてその上限を変化させ，それによる予測性能の変化を調べる．図 2-10 は DS1，DS2 について，2.3.5 節に述べたインパクトスケール有りの最良モデルを用いて探索上限距離（横軸）に制約を課す変更を加えたインパクトスケールで評価した予測性能 ddr_{10}（縦軸）である．探索上限距離が 1 ならば，インパクトスケールは定義上 Fan-in と Fan-out の和にほぼ等しいため，上限距離が 1 で ddr_{10} が十分大きければ，遠隔ノードを考慮することに意義が無いことになる．しかし，図では，DS1 では距離 5 まで，DS2 では距離 3 まで，ddr_{10} が増加し続けており，遠隔ノードの考慮に意義があることを示している．また，考慮すべき上限距離もソフトウェア毎に異なることを示している．ただ，DS1，DS2 ともに上限距離をある程度以上に大きくしても ddr_{10} に大きな影響を与えないことから，上限距離は単に十分に大きく取ればよいと言える．

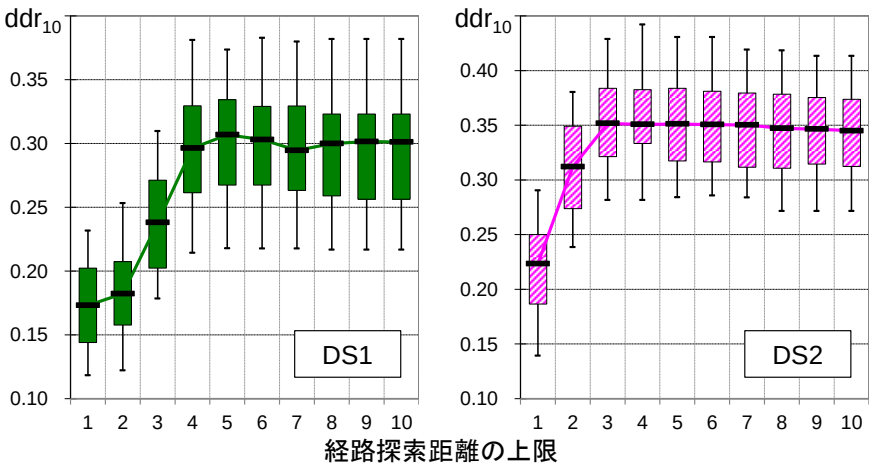


図 2-10　遠隔ノードを考慮することの効果

次に，「関係依存的な波及」を組み込んだことに意義があるかを調べる．図 2-11 の箱ひげ図中のモデル MET+IS は，2.3.5 節に述べたインパクトスケールを含んだ最良モデル，モデル MET はインパクトスケールを含まない最良モデルである．モデル NoCut はカットルールを除くことで，関係依存的な波及を用いないよう定義を変えたインパクトスケールを含む最良モデルである．三者の ddr_{10} を比較すると，モデル NoCut はモデル MET に対し極めて僅かな向上しかもたらさない．よって，カットルールの無い定義には意義が無いと言える．逆に言えば，カットルールの存在がインパクトスケールと欠陥との相関

を高めることができたと言える．ddr_{20} と AUC についても図 2-10 と図 2-11 と同様の結果が見られた．

以上により，確率的な波及により遠隔ノードの影響をモデル化し，そこに関係依存的な波及を導入することで得られた影響波及量であるインパクトスケールは，欠陥予測の性能を高めることができたと言える．正確な影響波及解析は実践的場面では実行困難なため，その結果に基づいた欠陥予測を行うこともまた困難である．インパクトスケールを用いた欠陥予測は，そのような解析に基づく予測を代替として十分に機能することが分かった．結論として，仮説 2 は本研究において妥当と言える．

仮説 2（再掲）： 確率的かつ関係依存的な波及を持つ波及モデルは，正確だが計算量やデータ収集コストが大きい解析の代わりとなるのに十分な予測性能を持つ．

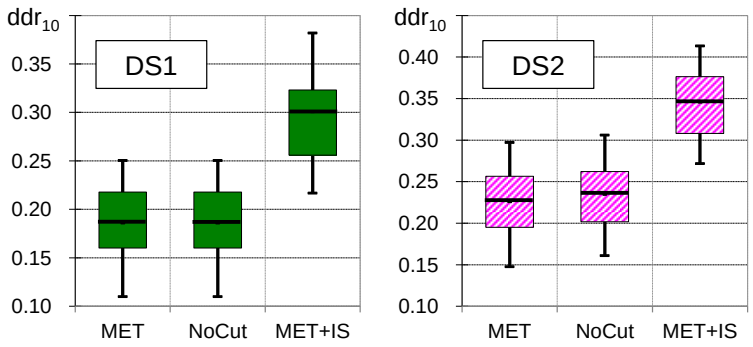


図 2-11　関係依存的な波及の効果

2.4.3 一般化のための留意点

本節では，本章の提案や評価結果を広く一般化する際に課題となること，留意すべきことを挙げる．インパクトスケールの定義は言語非依存である．とは言え，評価対象システムの記述言語である COBOL は他言語とは流儀もパラダイムも異なるため，本章の結論をそのまま他の言語に拡大するには配慮を要する．例えば，本章では欠陥予測の単位を「プログラム」（関数相当）としたが，他の言語では欠陥予測の単位は複数の関数からなるクラスやソースファイルであることが多いだろう．また，対象システムは勘定系システムの領域に属するが，一般化のためには他の領域のシステムの評価も必要となる．

本章の評価では，インパクトスケールの他に採取したメトリクスが 7 つと少ない．対象言語がオブジェクト指向言語であれば既存の多くのオブジェクト指向メトリクスがあり[2-35]，それらの集合にインパクトスケールを加えたその上で意義があるかは検証を要する．

また，IS の測定にはコールグラフなど依存関係の抽出が必要である．もし抽出した情報の精度が低ければ，インパクトスケールの測定値は影響範囲量を正確には反映しな

い．インパクトスケールは要約統計量なので抽出精度に対し過敏ではないと期待されるが，本章では動的束縛の少ない COBOL 言語で書かれたシステムを評価対象とすることでこの問題を極力避けたため，この影響については未評価である．

2.5 関連研究

ソフトウェアのコードとその関係からなるソフトウェア構造メトリクス[2-2]の既存のものには Fan-in，Fan-out，LCOM，CBO[2-35]がある．これらは依存グラフの隣接ノード間の直接的な関係のみを考慮するものだったが，本章の評価ではグラフ上の間接的な遠隔ノードも欠陥に強く影響することが判った．よって，遠隔ノードの情報を欠陥予測に組み入れるメトリクスは有効と言える．

変更量に関するプロセスメトリクスは欠陥予測の説明変数として有効である[2-4][2-6][2-7]．インパクトスケールがそれらのプロセスメトリクスとは独立の欠陥要素であれば価値が高い．Cataldo ら[2-11]は変更量と論理的結合関係が相互に独立でかつ両方が欠陥数と相関があることを報告した．論理的結合関係とインパクトスケールは依存関係と影響波及といった共通要素を持つため，インパクトスケールも変更量メトリクスと独立であると推測される．

Geiger らはコードクローンと論理的結合関係の間にある程度の関係があると報告した[2-36]．コードクローンはコールグラフやデータ依存のみではすべてを捕捉することはできないため，論理的結合関係のすべてをインパクトスケールで捕捉することはできない．コードクローンの関係をインパクトスケールの波及グラフに加えることは容易であり（例えば，新しい関係タイプ CLONE を追加），そのような情報の追加により予測性能の向上に寄与する余地がある．

近年は依存グラフを探索するメトリクスが現れてきている．井上らは，ソフトウェアコンポーネントの利用依存グラフをマルコフ連鎖モデルとみなすことで，コンポーネントの利用性の重要度を求めるコンポーネントランク[2-37]を提案している．早瀬らは，影響波及解析により構築した依存グラフを探索して保守工数の推定量を得る保守ポイントを提案している[2-38]．保守ポイントは評価値がグラフ経路上で漸減する点はインパクトスケールと共通するが，インパクトスケールは関係依存的な波及を導入していることが差異である．影響波及が確率的に起こるという仮定の下に，変更の予測を行うモデルの研究もある[2-18][2-19]．Tsantalis らのモデル[2-19]は変更履歴を用いてモジュールの変更確率を予測する．このモデルは，各変更が波及によるものか起源的なものかの教師情報を人間が与える必要があるため，スケーラビリティに課題がある．

ネットワーク尺度の導入によって欠陥予測性能が向上するかについては Zimmermann と Nagappan の提案[2-12]以後，主成分回帰を用いて追試が行われている[2-29][2-30][2-31]．Tosun ら[2-29]は小規模ソフトウェアと大規模ソフトウェアを対象に追試し，大規模では有効だが小規模では効果が無いと報告した．主成分分析により多数の説明変数から合成される「主成分」は人間にとって解釈が難しいという欠点を持つ．一方，インパクトスケールは単一のメトリクスでありながら，欠陥と相関があり，解釈が容易という利

点を持つ．また，ネットワーク尺度ではデータ依存などを数値に反映できないが，インパクトスケールは関係依存の波及を備えることによりネットワーク尺度では扱えない欠陥要素を扱うことができ，欠陥予測性能の更なる向上を可能にした．

2.6 まとめ

本研究では，影響波及の大きさが大規模ソフトウェアにおける欠陥発生の要因の一つであると仮定して，影響波及量を定量化したメトリクス「インパクトスケール」(ImpactScale，IS)を定義した．インパクトスケールは波及の有無を確率的に扱い，関係依存的な波及探索を行うモデルを用いることを特徴とし，大規模ソフトウェアでも現実的な時間で測定でき，直観的な解釈が可能という性質を備える．

2 つの大規模企業システムを対象に，インパクトスケールを欠陥予測モデルに追加することで予測性能が向上するかを，適合率・再現率・F1 値と工数考慮モデルの二つの基準で評価した．工数考慮モデルは，欠陥と予測されたモジュールに必要な工数を考慮するという実践的な観点で予測性能を評価する手法である．すべての評価において，インパクトスケールを含む予測モデルは，含まないモデルに比べて欠陥予測性能が高いという結果が得られた．例えば，インパクトスケールを既存のプロダクトメトリクスに追加することで，10%検査工数において 50%以上の欠陥検出数の向上が見られた．また，既存のプロダクトメトリクスとネットワーク尺度を併せて用いた予測モデルにさらにインパクトスケールを追加した場合にも予測性能が向上した．以上の結果は，既存の多くのメトリクスがありながら，敢えて新しいメトリクス「インパクトスケール」を定義することの有用性を示している．

著者らは既に，インパクトスケールを用いた欠陥予測による品質管理を実施しており，例えばある顧客の事例では，予測欠陥密度が上位のモジュールに集中して年間予算の 1/20 に当たる工数でレビューを施したところ，年当たりの欠陥発生が数件程度のシステムにおいて，8 件の欠陥を検出することができ，少ない工数で有効な予防措置を講じることができた．野中ら[2-39][2-40]は，複数のネットワーク機器製品のソフトウェアにインパクトスケールを適用した欠陥予測を実施し，予測精度が向上したことを示し，影響波及量を抑える変更が実務上有効であるとの知見を得た．

今後の課題としては，Java など他言語での効果の実証を行うとともに，インパクトスケールの用途として意図されている工数予測，品質劣化の監視などにも応用を広げ実証していく予定である．

3. 工数予測の精度向上のため対数変換と補正方法

ソフトウェア開発工数予測において，線形重回帰モデルは最も基本的な予測モデルとして多くの採用実績がある．その適用の前処理として，変数変換（特に，対数変換）が有効であるが，その理論的根拠は必ずしも明らかでなかった．本研究では，対数変換を行った線形回帰モデルは，指数曲線モデルと等価であり，ソフトウェア開発データの特徴を現すのに適していることを示す．ただし，対数変換を行ってから線形回帰を行うと，逆変換する際に過小予測するバイアスが発生してしまうことは見過ごされがちである．本研究ではその補正方法も示す．

3.1 はじめに

本章では，ソフトウェア開発工数（人月または人時）の予測に線形重回帰モデルを用いる場合を取り扱う．一般に，線形重回帰モデルは式(3.1)の形式を持つ．

$$\hat{Y} = \sum_{j=1}^{n} k_j N_j + C \tag{3.1}$$

ここで，$\hat{Y}$は目的変数であり開発工数の予測値である．N_jは説明変数（プロジェクト特性），k_jは偏回帰係数，Cは定数項である．

この式では，ソフトウェア開発工数が，プロジェクト特性（開発規模，期間，言語，アーキテクチャなど）の線形結合により表現される．線形回帰モデルの予測性能を高める方法として，前処理に変数変換（特に，対数変換）を行うことが知られているが，その意義や理論的根拠も必ずしも明確でない．近年，Kitchenham と Mendes[3-6] は，変数変換の重要性を指摘し，目的変数である開発工数に加えて，規模の尺度（ファンクションポイント）を対数変換すると，値の分布が正規分布に近づき，予測性能のよい線形回帰モデルができることを示している．ただし，線形回帰の適用条件として，説明変数が正規分布していることが求められているわけではない[3-8]．また，対数変換を行ってから線形回帰を行うと，各プロジェクト特性は工数に対して（加法的でなく）乗法的に作用するようになる．この点についても，従来，明確な説明が行われていない．

そこで，本研究では，線形回帰モデルを構築するにあたって，対数変換を行うことの意義や理論的根拠を明らかにする．

以降，3.2 節では，対数変換を行った線形回帰モデルは，指数曲線モデルとほぼ等価であり，現実をより正しく表すモデルとなっていることを示す．3.3 節では，プロジェクトデータの特徴から，対数変換の妥当性を論じる．3.4 節では，ケーススタディを通して，対数変換の有効性を示す．3.5 節はまとめである．

3.2 対数変換を伴う線形重回帰モデル

3.2.1 工数見積もりモデルの一般形

対数変換の意義を明らかにするために，本節では，多くの企業で用いられている工数見積もりモデルの一般形を紹介し，対数変換してから線形回帰を行った式と同じ形となることを示す．工数見積もりモデル(コストモデルとも呼ばれる)の一般形は，式(3.2) のようなべき関数で表される[3-11].

$$E = sL^c \tag{3.2}$$

ここで，E は開発工数，L は開発規模，s は生産性調整係数である．

COCOMO などの代表的なコストモデルにおいてもべき関数が採用されており，適用事例も多い．初級 COCOMO では，開発形態によって s，c の値が決まり，小規模開発を想定した organic モードでは $s = 2.4$，$c = 1.05$ である[3-1]．ここで，式(3.2)の両辺を対数変換*すると，

$$\log(E) = c \cdot \log(L) + \log(s) \tag{3.3}$$

となり，実は，規模 E と工数 L を対数変換してから線形回帰を行った場合と同じ形となる．このことから，対数変換してから線形回帰を行うことは，工数と規模の関係をモデル化するのに都合がよい．

同様に，開発期間と工数の関係についても，べき関数への当てはまりが良いことが知られており，開発期間は工数の概ね 3 乗根に比例するという経験則がある[3-1]．従って，対数変換してから線形回帰を行うことで，工数と開発期間の関係についてもうまくモデル化できる．逆に，対数変換を行わない naïve な線形回帰モデルは，規模，工数，開発期間の関係を表すには本来向いていないといえる．

3.2.2 対数変換による指数曲線モデルの構築

本章では，線形回帰の前処理として，全ての説明変数および目的変数を対数変換する場合を想定する．得られるモデル式は，式(3.4)の通りである．このモデル式を本章では，log-log 重回帰モデルと呼ぶことにする．

$$\log(\hat{Y}) = \sum_{j=1}^{n} k_j \log(N_j) + C \tag{3.4}$$

ここで，式(3.4)の両辺について exp を取ると，式(3.5) の指数曲線モデルが得られる．従って，log-log 重回帰モデルは，指数曲線モデルであるとも言える．

---

* 本章では自然対数を用いる．

$$\hat{Y} = \exp(C)\prod_{j=1}^{n} N_j^{k_j} \tag{3.5}$$

式(3.5)で明らかとなったように，一見，非線形回帰に見える指数曲線モデルが，対数変換によって単純な線形回帰で得られるのである．式(3.4)の log-log 重回帰モデルから予測工数$\hat{Y}$を得るためには，出力値log $(\hat{Y})$の exp を取る必要があり，このことは，式(3.5)の指数曲線モデルから予測値を得ていることと等価である．

指数曲線モデルでは，各プロジェクト特性は工数に対して（加法的でなく）乗法的に作用する．このことは，3.4 節で後述するように，開発の生産性に寄与する説明変数を含む場合にモデルの当てはまりが良い．

3.2.3 指数曲線モデルの残差

式(3.5)の指数曲線モデルは，残差にバイアスを生じる（過小予測傾向となる）という落とし穴がある．以下，指数曲線モデルの残差について述べる．プロジェクト i の工数を Y_i，その残差をϵ_i，プロジェクト特性を $N_{i,j}$ とおくと，log-log 重回帰モデルにおける各変数と残差の関係は，次式で表される．

$$\log(Y_i) = \sum_{j=1}^{n} k_j \log\left(N_{i,j}\right) + C + \epsilon_i \tag{3.6}$$

線形回帰モデルが妥当となるためには，この残差が以下の性質を満たすことが仮定される[3-5][3-8][3-10].

- 独立性：ϵ_iとϵ_kは互いに独立である（$i \neq k$）.
- 不偏性：ϵ_iの期待値はゼロである.
- 等分散性：ϵ_iの分散は全て等しい.
- 正規性：ϵ_iは正規分布に従う.

ここでは，このϵ_iが従う正規分布を $N(0, \sigma)$とする.

一方，式(3.6)を変形すると，指数曲線モデルにおける測定値Y_iと予測値$\hat{Y}_i$の関係は，

$$\begin{aligned} Y_i &= \exp(\epsilon_i)\exp(C)\prod_{j=1}^{n} N_{i,j}^{k_j} \\ &= \exp(\epsilon_i)\hat{Y}_i \end{aligned} \tag{3.7}$$

となり，その残差 ζ_i は，

$$\begin{aligned} \zeta_i &= Y_i - \hat{Y}_i \\ &= \hat{Y}_i(\exp(\epsilon_i) - 1) \end{aligned} \tag{3.8}$$

となる．以降では，log-log 重回帰モデルでの残差ϵ_iの性質が，指数曲線モデルでの残差 ζ_i にどのように現れるかを述べる.

まず，独立性が保たれることは，$i \neq j$ のときϵ_iとϵ_jが独立ならば，$\log(\epsilon_i)$，$\log(\epsilon_j)$ も独立であることから明らかである．

次に，等分散性については，残差 ζ_i が$\hat{Y}_i$に比例する値となり，これは工数見積もりモデルとしてむしろ好都合である．式(3.8) は，工数が小さなプロジェクトは予測誤差が小さく，大きなプロジェクトになるほど予測誤差が大きくなることを意味する．一般に，大規模プロジェクトほど工数の見積もり誤差も大きくなるため，このような残差の振る舞いをする指数曲線モデルは，現実をモデル化するのにより都合がよい．一方，naïve な線形回帰モデルでは，プロジェクトサイズに応じた分散の変化をうまくモデル化できない．

次に，正規性については，残差ζ_iは正規分布ではなく-1 だけシフトした対数正規分布となり，プラス側に裾野の広い分布となる．これは，工数の見積もり誤差は超過方向に大きく外れることが多いという現実に即していると言える．

最後に，不偏性についてであるが，式(3.8)における $\exp(\epsilon_i)$ は対数正規分布に従うため，その期待値は $\exp(\sigma^2/2)$となる．従って，残差 ζ_i の期待値は $\hat{Y}_i(\exp(\sigma^2/2) - 1)$となり，これは常に 0 よりも大きくなる．すなわち，このモデルは平均残差が常に正となる過小予測を起こしやすい偏ったモデルであり，これは実用上の問題となる．

3.2.4 残差のバイアスの補正

前節で述べたように，log-log 重回帰モデルを変形して得られる指数曲線モデルは，過小見積もりとなる傾向を持つ．ところが，従来，ソフトウェア工学分野では，残差にバイアスが生じることも，その補正方法についても，ほとんど論じられていない．本節では，Finney らの提案する補正方法[3-3][3-7][3-9] を述べる．E[・]を期待値を表す関数とする．式(3.7) より，$Y_i/\hat{Y}_i = \exp(\epsilon_i)$ であるため，次式が成り立つ．

$$\mathrm{E}[Y_i/\hat{Y}_i] = \mathrm{E}[\exp(\epsilon_i)] = \exp\left(\frac{\sigma^2}{2}\right) \tag{3.9}$$

σ は残差ϵ_i が従う正規分布の標準偏差であるが，それ自体は未知であるため，式(3.10)に示す σ の不偏推定量である SEE（standard error of estimate; 推定値標準誤差）で代替する．

$$\mathrm{SEE} = \sqrt{\frac{\sum_i \left(\log Y_i - \log \hat{Y}_i\right)^2}{r - p - 1}} \tag{3.10}$$

式(3.10)の分母の $r - p - 1$ はモデルの自由度である．r はモデル構築に用いたプロジェクトの件数であり，p は説明変数の数である．この自由度の調整はしばしば見落とされがちである[3-9]ため，注意が必要である．

式(3.9) と式(3.10) から，補正係数 CF(correction factor)が次式で与えられる．

$$\mathrm{CF} = \exp\left(\frac{(\mathrm{SEE})^2}{2}\right) \tag{3.11}$$

バイアスが補正された予測値$\hat{Y}^*$ は次式で与えられる．

$$\hat{Y}^* = \mathrm{CF} \cdot \hat{Y} \tag{3.12}$$

3.3 プロジェクトデータの特徴からの考察

本節では，プロジェクトデータの特徴から，対数変換の妥当性を論じる．Kitchenham と Mendes[3-6] は，ソフトウェア開発プロジェクトデータの多くは，工数と規模の関係に着目すると，(1) 規模が大きくなるほど工数のばらつきが大きくなる，(2) 規模の小さい部分にプロジェクトが集中している，(3) 外れ値がある，(4) 異なるタイプのプロジェクトが混在している，といった特徴があると述べている．以降では，これらの特徴と対数変換の関係について考察する．

3.3.1 規模が大きくなるほど工数のばらつきが大きくなる

多くのプロジェクトデータは，規模が大きくなるほど工数が大きくなり，それに伴って工数のばらつきも大きくなる．3.2.3 節で述べたように，log-log 重回帰モデルを式変形して得られる指数曲線モデルでは，残差が目的変数に比例して大きくなるため，この特徴をうまくモデル化でき，予測性能の向上が期待できる．

3.3.2 規模の小さい部分にプロジェクトが集中している

多くのプロジェクトデータは，規模の小さい部分にプロジェクトが集中しており，すそ野の広い分布となる．この分布は，対数正規分布に近いため，対数変換を行うことで，正規分布に近くなる．またその結果，線形回帰モデルの予測性能が向上することが多い．ただし，線形回帰の適用条件として，説明変数が正規分布していることが求められているわけではない[3-8]．あくまでも，経験的な Tips として，すそ野の広い分布を持つ変数は，対数変換を試してみると良い，ということである．

3.3.3 外れ値が存在する

一般に，外れ値とは，分布の中心から大きく外れた値のことを指す．3.3.1 節で述べたように，多くのプロジェクトデータは，規模が大きくなるほど工数のばらつきが大きくなり，外れ値も増える．そのため，残差が目的変数に比例して大きくなる指数曲線モデルの方が，対数変換を行わない naïve な線形回帰モデルよりも，現実をうまくモデル化できると考えられる．

3.3.4 異なるタイプのプロジェクトが混在している

ここでは典型的な例として，図 3-1 のように，規模と工数の関係が異なる 2 つのタイプのプロジェクトが混在している場合を考える．

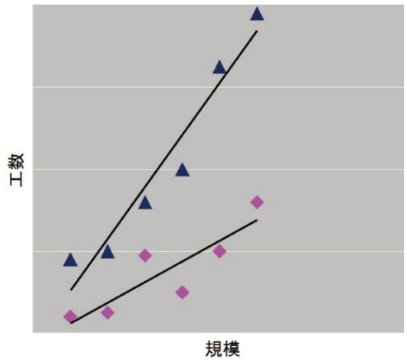


図 3-1　異質集団を含むデータセット

このような場合，プロジェクトのタイプが 2 値変数で与えられており，かつ，指数曲線モデルを用いた場合に，予測精度の向上が見込まれると考えられる．3.2.2 節で述べたように，指数曲線モデルの一つの特徴は，各説明変数が，加法的ではなく乗法的に結合されるため，回帰曲線の傾きに影響する要因をうまくモデル化できるためである．例えば，アーキテクチャや開発言語など，ソフトウェア開発の生産性に影響を与えている要因を説明変数に用いる場合に，特にうまくモデル化できる．

3.4 ケーススタディ

3.4.1 概要

線形回帰における対数変換の効果を分かりやすく示すケーススタディとして，実データ（Desharnais データセット[3-2]）を用いた残差分析と工数予測の例を示す．Desharnais データセットは，カナダのあるソフトウェア企業の開発実績データであり，無償で一般公開されているため，追実験が可能である．

本ケーススタディでは，過去の実績データを用いて将来のプロジェクトの工数予測を行うことを想定し，モデル構築用のフィットデータとして 1986 年以前のプロジェクト 58 件を用い，モデル評価用のテストデータとして 1987 年以降の 19 件を用いた．目的変数は開発工数である．説明変数は，調整済みファンクションポイント，開発期間，開発チーム経験年数，プロジェクトマネージャ経験年数，開発言語を用いた．開発言語については，２値変数化した．

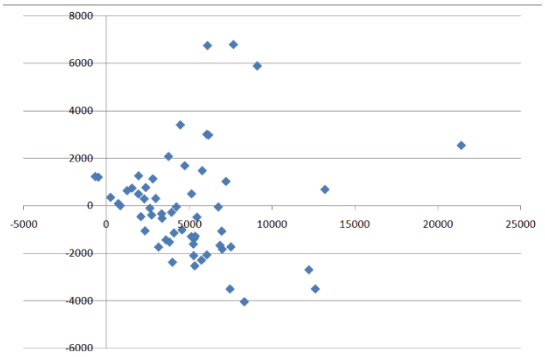


図 3-2　線形重回帰モデルの残差分布

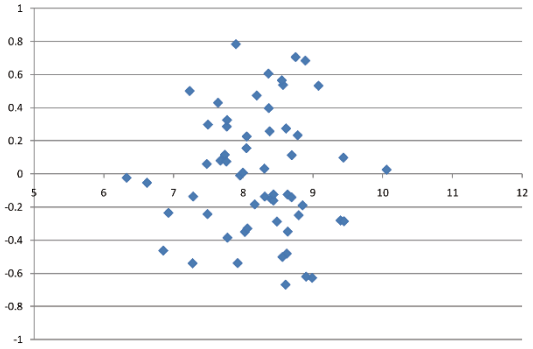


図 3-4　Log-log 重回帰モデルの残差分布

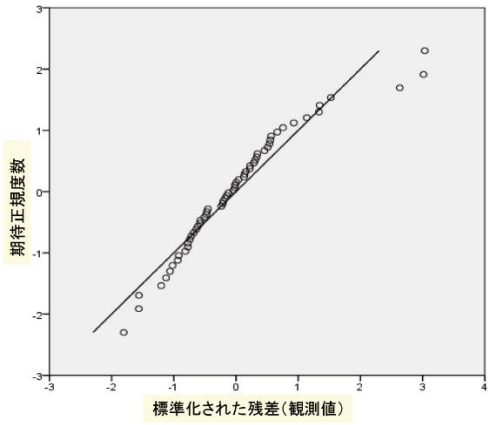


図 3-3　線形重回帰モデルの残差の正規 Q-Q プロット

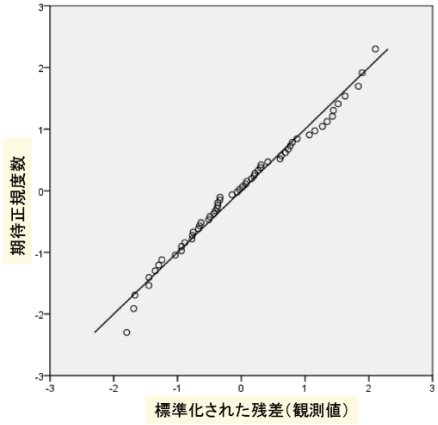


図 3-5　Log-log 重回帰モデルの残差の正規 Q-Q プロット

3.4.2 モデルの残差

Naïve な線形回帰モデルの残差の散布図を図 3-2 に，残差の正規 Q-Q プロットを図 3-3 に示す．図 3-2 では，回帰式による開発工数の予測値$\hat{Y}$を横軸に，対応する残差の値を縦軸にとっている．図 3-2 から，開発工数が 0 に近いところでは残差が小さく，開発工数が大きくなるにつれ残差が大きくなっており，残差の等分散性が満たされていないことが分かる．また，図 3-3 より，各プロットは直線からやや外れており，残差の正規性についても満たされているとはいえない．このことから，この線形回帰モデルは，本データセットをモデル化するのに適していないといえる．

次に，log-log 重回帰モデルの残差の散布図を図 3-4 に，残差の正規 Q-Q プロットを図 3-5 示す．図 3-4 より，開発工数の大小にかかわらず残差はほぼ同じ幅で分布しており，残差の等分散性がほぼ満たされていることが分かる．また，図 3-3 と比較すると，図 3-5 では各プロットは直線上に乗っており，残差の正規性についてもほぼ満たされているといえる．

Naïve な線形回帰モデル，（log-log 重回帰モデルを変形して得られる）指数曲線モデル，および，3.2.4 節の方法により残差の補正を行った指数曲線モデルの残差平均を表 3-

1 に示す．表中，MMRE (Mean Magnitude of Relative Error)，MMER (Mean Magnitude of Error Relative) はいずれも相対誤差平均の尺度であり，MMRE = $\mathrm{E}[|Y_i - \hat{Y}_i|/Y_i]$，MMER = $\mathrm{E}[|Y_i - \hat{Y}_i|/|\hat{Y}_i|]$である．また，表中の MR は Mean Residual（残差平均）であり，線形回帰では MR が 0 になるように偏回帰係数が定められる．

表 3-1　各モデルの残差

	MMRE	MMER	MR
線形回帰モデル	0.426	0.404	(0)
指数曲線モデル	0.313	0.324	291
指数曲線モデル（補正）	0.347	0.309	-88.4

表 3-2　各モデルの予測誤差

	MMRE	MMER	MR
線形回帰モデル	0.809	0.784	-242
指数曲線モデル	0.280	0.295	327
指数曲線モデル（補正）	0.289	0.284	5.61

表 3-1 より，MMRE，MMER のいずれの指標においても線形回帰モデルよりも指数曲線モデルが優れており，予測性能がよいことが示唆される．また，MR に着目すると，補正前は MR=291 であり，実測値に対して平均で約 5.8%の過小予測をするモデルとなっていた．補正後は MR=-88.4 と大幅にバイアスが打ち消されている．相対残差について補正後を補正前と比べると，MMRE が少し悪化した一方で，MMER が少し改善している．従来，MMRE は過小予測傾向のモデルを不当に良いと評価してしまうことが知られており[3-4]，残差の補正によって過小予測傾向が解消されたため，MMRE が増大したと考えられる．

3.4.3 モデルの予測性能

モデル評価用のテストデータに対する予測を行った結果を表 3-2 に示す．表 3-2 より，naïve な線形回帰モデルは，指数曲線モデルと比べて，すべての指標において予測精度が大きく低下していることが分かる．また，MR に着目すると，補正前は明らかなバイアスを持つ問題があるが，補正後はそのバイアスがほとんど打ち消されている．予測誤差について補正後を補正前と比べると，MMRE，MMER は大差ないのでバイアス補正のデメリットはほぼ無い．

モデル構築時の残差平均（表 3-1）と誤差平均（表 3-2）を比べると，表 3-2 では，指数曲線モデルの優位性は，さらに大きくなった．このことは，誤ったモデル化を行うことの危険性を示唆している．データセットが当該モデルの仮定に合わない場合，構築し

たモデルが一定の残差平均を示したとしても，予測時の誤差が顕著となる場合があることを示唆している．

3.5 まとめ

本章では，対数変換を行った線形回帰モデルは，指数曲線モデルと等価であり，ソフトウェア開発データの特徴を現すのに適していることを示した．また，得られた指数曲線モデルは，残差にバイアスを生じ，過小見積もりとなる傾向を持つため，その補正方法も示した．ケーススタディでは，構築したモデルの残差分析，及び，残差平均の導出により，naïve な線形回帰モデルよりも指数曲線モデルがより現実をうまく表しており，予測精度も良いことを示した．

対数変換の手続きはごく簡単で効果も大きいため，線形回帰を行う場合には，対数変換を試してみることが望ましいといえる．

4. 依存関係グラフを用いたソフトウェアアーキテクチャの復元

4.1 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[4-1]．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（本研究での「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[4-1]，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[4-2]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．

本研究で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」（Software Architecture Finder 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（つまり，非機能要件を含まない）機能」[4-3]という意味で用いる．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．

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[4-2][4-4][4-5][4-6]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[4-1][4-7][4-8]．介入

を要する大きな要因が「遍在モジュール(omnipresent module)」[4-9]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[4-2]．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[4-9]，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[4-2][4-10][4-11][4-12]．遍在モジュールの自動除去についてはいくつかの提案[4-4][4-9][4-13][4-14]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，遍在モジュール除去作業を不要とすることで自動化を実現する．

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[4-15]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，[4-16]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．本章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，[4-16]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[4-15][4-17]を用いていたが，より優れるとされる Louvain 法[4-18]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．

本章の以降の構成は次の通りである．4.2 節にて関連研究を示し，4.3 節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，4.4 節では実験計画について述べ，4.5 節でケーススタディを示し，4.6 節で実験結果を示し，議論する．4.7 節にて妥当性の脅威について述べ，4.8 節にてまとめる．

4.2 関連研究

4.2.1 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 [4-2]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 [4-4]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[4-19]．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[4-20][4-21]．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[4-22]やクラスタリングを行う手法[4-12]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[4-10][4-21][4-23]，k-means などの非階層クラスタリングを用いる手法[4-20]，パターンマッチングを用いる手法[4-4]，グラフクラスタリングを用いる手法[4-2][4-5][4-7][4-24]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[4-24]．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[4-14]，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[4-7]，対話的遺伝アルゴリズムを用いるもの[4-8]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[4-25]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[4-5][4-24]．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[4-15]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[4-6].

4.2.2 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[4-26]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 4-3）の合計数である．MoJo の欠点を改善した指標 MoJoFM が提案され[4-27]，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 [4-28]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[4-29]，複数の指標を用いた評価の枠組みを提案した[4-30].

Wu ら[4-23]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[4-20]．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[4-2][4-4][4-11][4-21]

など 9 つの手法の比較を行った[4-31]．本研究の評価はこれら既存の枠組みに則って行う．

4.3 SArF ソフトウェアクラスタリングアルゴリズム

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．

4.3.1 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 4-1 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 4-1 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．

$$\mathrm{D}(A,B) = \frac{1}{\mathrm{fanin}(B)} \tag{4.1}$$

図 4-1 の例の各専念度は D(A,X) = 1，D(B1,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．

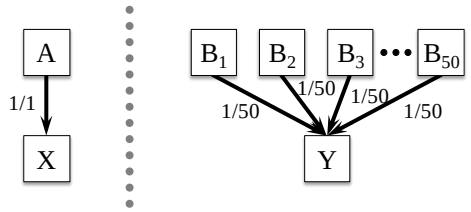


図 4-1　専念度スコアの計算例（単純な場合）

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 $D_M(A,B)$の定義は次式となる．

$$\mathrm{D_M}(A,B) = \sum_{m \in M_{B \leftarrow A}} \frac{1}{\mathrm{fanin_X}(m) \cdot \mathrm{m_X}(B)} \tag{4.2}$$

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．$\mathrm{m_X}(B)$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(4.2)を用いて計算される．式(4.2)の例を図 4-2 に示す．$\mathrm{m_X}(X)=3$，$\mathrm{fanin_X}(X.init)=1$，$\mathrm{fanin_X}(X.set)=1$，$\mathrm{fanin_X}(X.use)=3$ である．クラス A からクラス X への専念度 DM(A,X)=7/9 のうち，メソッド A.a からメソッド X.init への寄与は 1/3，A.a から X.set への寄与は 1/3，A.f から X.use への寄与は 1/9 である．

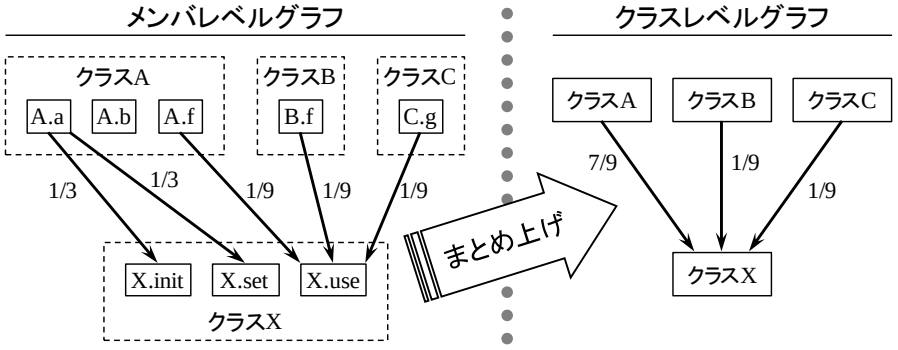


図 4-2　専念度スコアの計算例（階層関係がある場合）

4.3.2 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[4-15]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 [4-32]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[4-33].

$$\mathrm{Q_D} = \frac{1}{W}\sum_{i,j}\left[A_{ij} - \frac{k_i^{OUT} k_j^{IN}}{W}\right]\delta(c_i, c_j) \tag{4.3}$$

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(4.3)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(4.3)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 $\mathrm{Q_D}$ 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[4-34]が，良い近似を与える解法が存在している．著者らは[4-16]で Newman のアルゴリズム[4-15]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[4-17]．本研究では Louvain 法[4-18]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[4-35]．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[4-18]）．

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 $\mathrm{Q_D}$ の増加が最大になるクラスタへ移動する．
3. 2.の走査を $\mathrm{Q_D}$ 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，$\mathrm{Q_D}$ 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 4.6.3.6 節で調べる．

4.3.3 SArF アルゴリズム

本研究で提案する 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. 4.3.1 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. 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．

4.4 実験計画

本節では，4.5 節と 4.6 節で用いる性能指標と性能測定の方法について説明する．

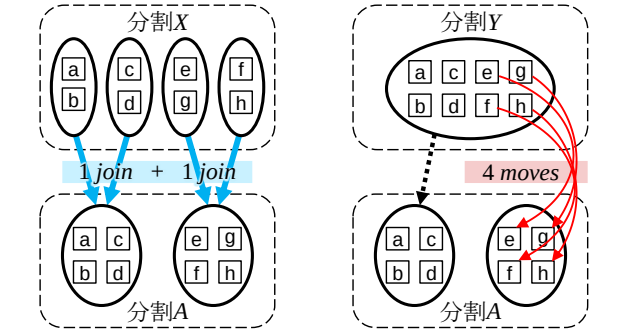
4.4.1 性能指標

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[4-23]を，批判[4-20]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．本研究でも[4-20][4-21]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，4.5 節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，4.6 節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バラバラ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．本研究では評価の妥当性を増すために，「平均的な基準」による評価に加え，自身の観点に沿ったケーススタディを行っている．また，[4-16]では追加のケーススタディを実施している．

オーソリティ準拠度の指標として MoJoFM[4-27]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 4-3 に示す．



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

図 4-3　move／join 操作と 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 4-3 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．本章を通し，*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[4-26]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% と定義される対称な類似度である．

実行時間は，図 4-4 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．

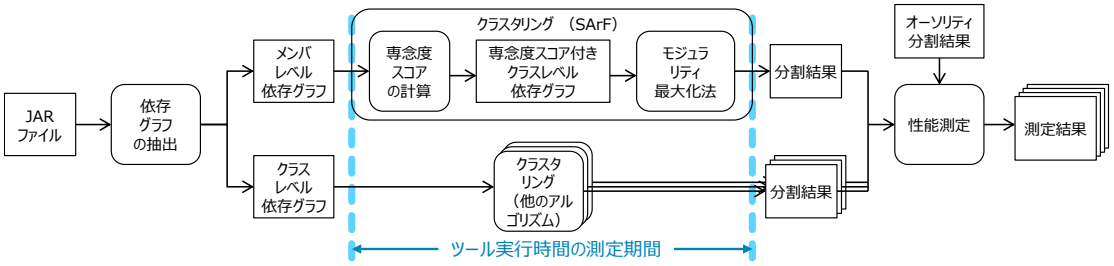


図 4-4　性能測定手順

4.4.2 性能測定手順

性能測定手順を図 4-4 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[4-36]．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 4-2 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．

4.5 ケーススタディ

SArF の出力の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[4-37]ため，既存研究でよく用いられる[4-6][4-12]．そのほとんどのパッケージがフィーチャーに一致する[4-12]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．[4-16]にはより詳細な解析と別のケーススタディがある．

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 4-5 は，文献[4-12][4-37]と JAR ファイルの情報を利用して作成した Weka のアー

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

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．

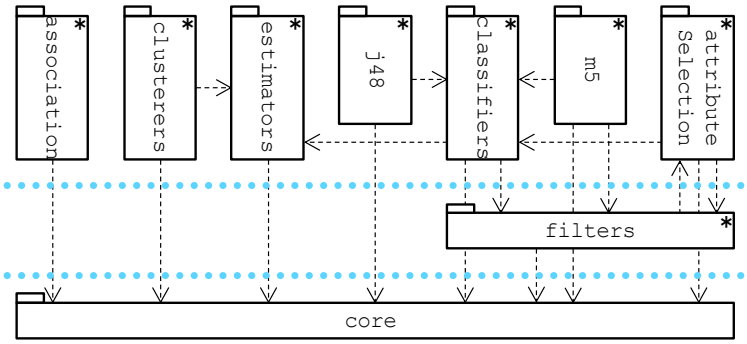


図 4-5　Weka 3.0 のアーキテクチャ（*が付されたパッケージはフィーチャーに一致）

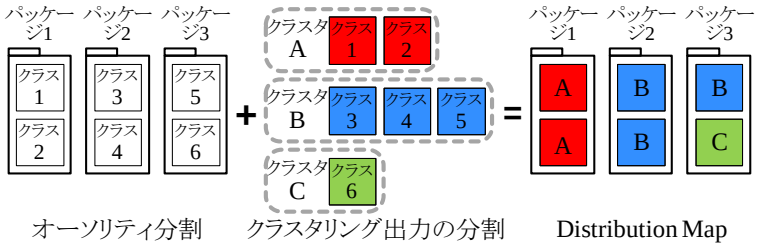
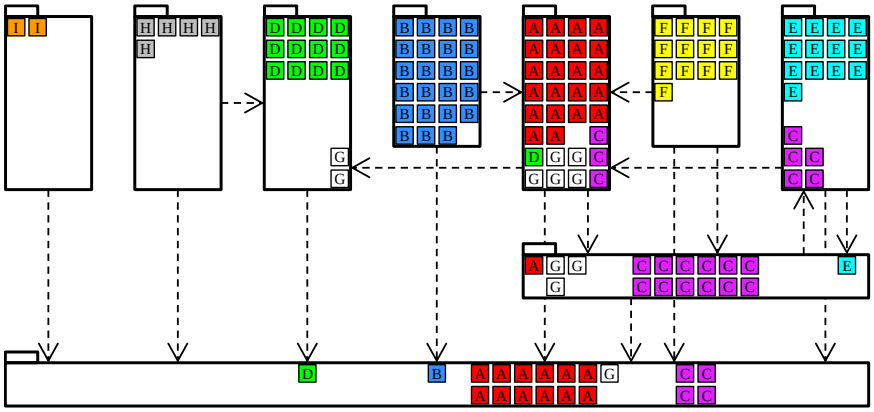


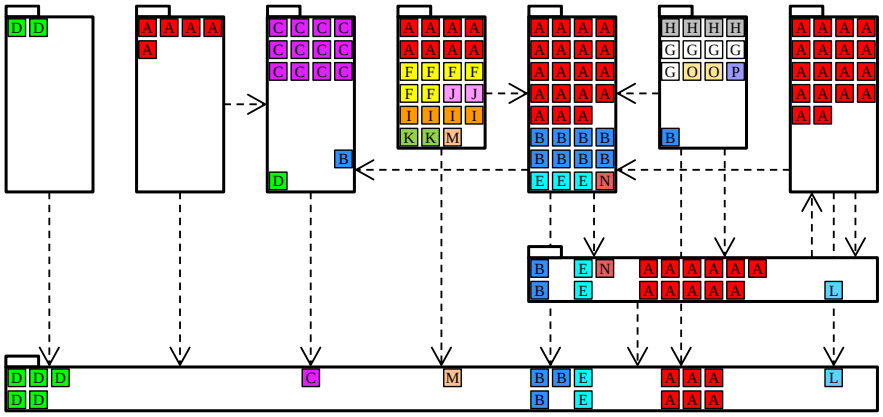
図 4-6　Distribution Map による可視化の例

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 4-6 に例を示す Distribution Map[4-38]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．

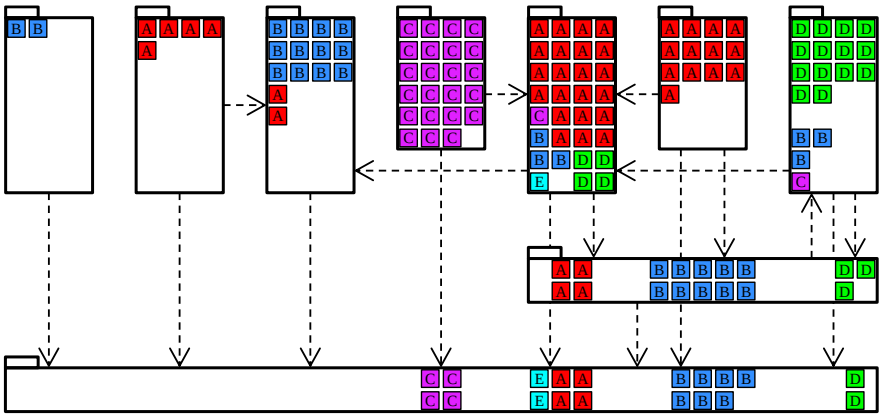
4.4.2 節の手順に従ってクラスタリングアルゴリズムを実行した結果を可視化した例を図 4-7 に示す．図 4-7(a)は SArF の結果である．各クラスタが core 以外のパッケージにほぼ一致していることが判る．これは，SArF が効果的にクラスを対応するフィーチャーに分割していることを意味し，MoJoFM 値が 72.9%と高いことに整合する．なお，クラスタリングアルゴリズムの評価に理想的なオーソリティ分割結果が得られることはほぼなく，その場合，最も正確な結果であってもオーソリティ準拠度は 100%には至らない[4-16]．[4-16]によれば，この 72.9%はほぼ上限に近い．



(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 4-7　Weka のクラスタリング結果の可視化

比較対象アルゴリズム ACDC と Bunch の結果を図 4-7(b)と図 4-7(c)に示す．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．

4-6 評価

本節では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[4-6][4-15]（以後 Newman と略），代表的な既存研究である ACDC[4-4]，Bunch[4-2]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．

4.6.1 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[4-16].

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．

評価対象とするバージョンは冗長性を省くためメンテ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．

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

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 4-1 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．

4.6.2 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 4.4.2 節の手順に従って性能測定を行った．結果を表 4-1，図 4-8，図 4-9，図 4-10 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 4-1 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 4-1 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 4-8 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 4-9 に *Ka* と *K* の散布図を示す．図 4-10 にクラス数と実行時間の散布図を示す．

4.6.3 議論

4.6.3.1 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 4-1 の MoJoFM の結果から，SArF が最良（太字）となることが多いが，ACDC と Newman が最良となることもある．MoJoFM の最良値はソフトウェアによって 40〜70 の値をとるが，MoJoFM には上限値が存在する[4-16]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 4-8(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 4-2 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 4-1 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いが，Newman と Bunch が最良となることもある．ACDC は他と比べて MRE の値が格段と大きい．表 4-2 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 4-9 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 4-8(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．

4.6.3.2 安定度

測定された安定度の分布を図 4-8(c)に示し，比較結果を表 4-2 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．

4.6.3.3 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．

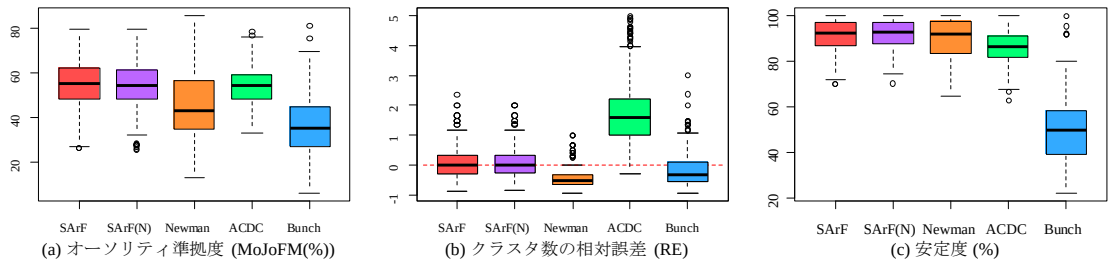


図 4-8　測定結果の箱ひげ図

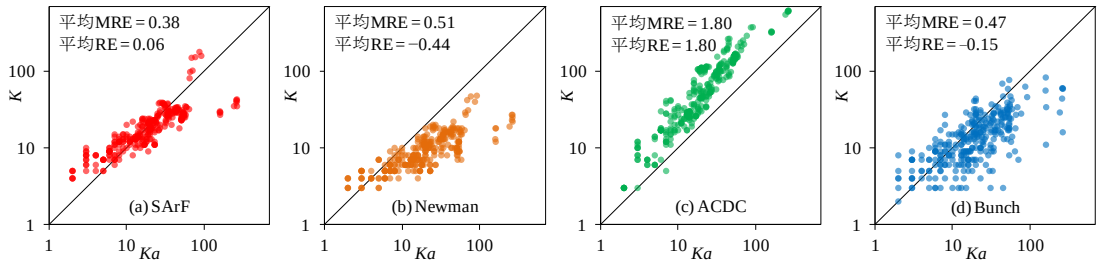


図 4-9　オーソリティ分割結果のクラスタ数 *Ka* とアルゴリズムの出力クラスタ数 *K* との比較

表 4-1　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）

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)

全ソフトウェア全バージョンの実行時間の散布図を図 4-10 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 4-3 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．

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．

4.6.3.4 SArF の有効性

4.6.3.1，4.6.3.2，4.6.3.3 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．

4.6.3.5 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[4-18]であり，Newman アルゴリズム[4-15]を用いる SArF(N)[4-16]との得失を調べる．図 4-8 より SArF の品質と安定度は SArF(N)と大差ないことが判る．表 4-4 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 4.6.3.3 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．

表 4-2　品質・安定度の比較

	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．

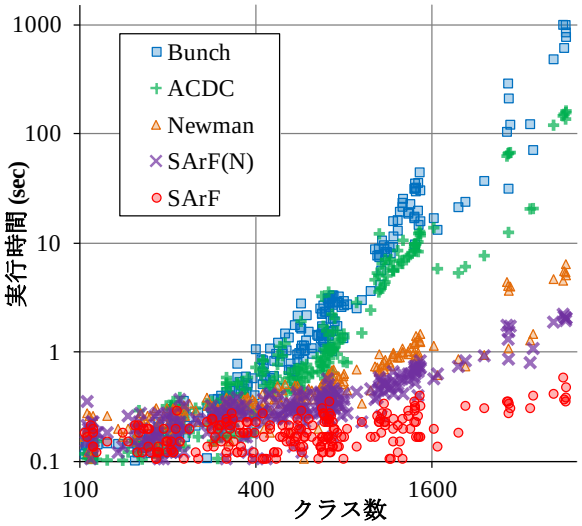


図 4-10　実行時間測定結果

表 4-3　実行時間測定結果（抜粋）

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 4-4　SArF と SArF(N)の比較

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

表 4-5　SArF とオリジナル Louvain 版 SArF の比較

	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

4.6.3.6 SArF とオリジナル Louvain 版 SArF との比較

4.3.2 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 4-5 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．

4.7 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．4.4.1 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，本章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，本章で評価対象アルゴリズムを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[4-16]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．

4.8 まとめ

本章で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[4-23]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．

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．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．

5. 依存関係グラフを用いたソフトウェアアーキテクチャの可視化

5.1 ソフトウェア保守におけるプログラム理解と可視化

ソフトウェアの保守において，ソフトウェアアーキテクチャを理解することは，重要な要素である．ドキュメントなどで維持されているソフトウェアアーキテクチャの知識は，往々にして現状と乖離することが多く，また知識自体が失われていることも多い．この知識を復元するために，様々なプログラム理解やソフトウェア可視化の手法が利用されてきた[4-1][5-1][5-2].

現在までに，パッケージやディレクトリなどの明示的なソースコード構造を可視化する研究は数多く存在する[5-3][5-4][5-5][5-6]．しかし，ソフトウェアアーキテクチャを深く理解するためには，ソフトウェアアーキテクチャの暗黙的な情報であるフィーチャーを明らかにする必要がある．ここで，「フィーチャー」という用語の定義として，フィーチャーロケーションの研究の文脈のものを採る．すなわち，フィーチャーとは，外部のユーザによって起動されることができるシステムの機能である[4-3]．このような暗黙的な情報を可視化するために，本章では新しいソフトウェア可視化手法である SArF Map を提案する．SArF Map は前章で提案した SArF (Software Architecture Finder)を利用しており，共通のフィーチャーを持つクラスを同じクラスタに集めることができる．SArF が出力したソフトウェアアーキテクチャの復元結果を，都市メタファー[5-7]を用いて，フィーチャーとレイヤーの観点から可視化する．都市メタファーは 5.2 節で詳しく述べるが，現実世界の知識を流用することで直観的に対象を理解させることができる利点を持つ．

フィーチャーはソフトウェアシステム上で抽象度の高い概念単位であるため，SArF Map のユーザは対象となるシステムについて高レベルの判断を下すことが可能であると期待される．また，開発者以外のステークホルダーにも解釈可能であり，様々な立場のステークホルダー間で対象システムについて共有可能なメンタルモデルとして利用することができる．また，SArF Map は依存関係グラフに基づいているため，可視化で得られた知識や洞察は，影響波及分析や再利用など，様々なソフトウェア開発・保守の活動に利用することができる．

SArF Map は，フィーチャーの観点に加えて，アーキテクチャ上のレイヤーの観点を可視化する．レイヤーとは，アーキテクチャを積み重なった複数の層として表現する場合の各層のことである．アーキテクチャをユーザ側・バックエンド側などの軸で整理するときに用いられる．この 2 つの異なる観点を可視化するために，SArF Map は 2 つの異なるレイアウト表現で構成されている．都市メタファーに従って，ひとつのフィーチャーをひとつの街区として表現し，レイヤーを北から南へと下る斜面上のビルの位置として表現している．これにより，フィーチャーをレイアウトすることは街区をレイアウトすることであり，レイヤーをレイアウトすることは街区内のビルをレイアウトすることである．このレイアウト手法を Street and Block Tree Layout と名付けた．フィーチャーの観点からのアーキテクチャ理解を容易にするため，関連するフィーチャーを表す街区を道で結び，可能な限り近い位置に配置する．このように，抽象的な概念を具体的な図として表現することで，現状把握が容易になり，保守されたドキュメントとの差異を比較す

るなどの作業が効率化される．

SArF Map は，基本となる SArF アルゴリズムが 4 章に述べた通り人手の介入が不要なよう自動化されているため，同様に自動化されており，その出力は決定的である．入力は対象ソフトウェアシステムの静的な依存関係グラフであり，言語依存性はない．これは例えば Java 言語ならば，jar ファイルから抽出が可能である．このように，SArF Map は高い適用可能性を持ち，利用が容易であるという特長を持つ．

5.2 関連研究

ソフトウェアの可視化は，大規模なソフトウェアシステムやそのアーキテクチャの概要を理解するためや[5-3][5-4][5-5][5-6]，機能など抽象的な概念を理解するため[5-8]など，様々な目的で研究されている．

都市メタファー[5-2]は，ソフトウェアの可視化に優れる多くの利点を持つため，多くの研究で用いられている[5-4][5-5][5-6][5-7][5-9][5-10]．ソフトウェア可視化ではアーキテクチャなどのソフトウェア全体を指す大域情報と，修正対象のソースコードを特定する局所情報との両方を同時に一貫性のある形で把握する必要がある．都市メタファーでは，都市と建物という現実世界のメタファーがこれに対応するため，ユーザが馴染みやすく，注目している局所位置を容易に把握できるという利点を持つ．また，現実世界の風景に情報を重ねることで様々なソフトウェア構造やメトリクスを同時に表現することができるのも利点である．学習時間が短いことも利点である．本提案でもこれら利点の恩恵をそのまま享受することができるため採用した．一方で，欠点もある．先入観により誤解を招きやすいこと，情報の配置に制約を受けるために表現が制約される場合があること，基本的に木構造やグラフ構造に基づく可視化になるため空間当たりの情報量の密度が疎になること，などである．Caserta と Zendra はまた，都市メタファーの欠点として，2 次元空間内にしかレイアウトできないことを指摘しているが[5-2]，本提案ではこの制約をむしろ活用している．2 次元空間上の白地図を，付加情報を持たないソフトウェアアーキテクチャの可視ビューとしてステークホルダー間で共有される基本のメンタルモデルとして利用する．

CodeCity [5-4][5-11]は，都市メタファーを用いたアプローチのひとつであり，クラスをビルに，パッケージを街区に見立て，ビルや街区を表す矩形を上位の矩形に詰めながら階層的に配置するパッキングツリーレイアウトアルゴリズムを用いてレイアウトする．様々なメトリクスを同時に表示したり，ソースコードの階層を自然に表現したりすることができ，高いスケーラビリティを持つ．SArF Map と，CodeCity など他の既存の都市メタファーの可視化手法の最も重要な違いは，SArF Map の街区は，ソフトウェアクラスタリングを使ってのみ抽出可能な暗黙の情報である「フィーチャー」であることである．

ソフトウェアクラスタリングとは，与えられたシステムをいくつかのサブシステムや管理可能なサイズのクラス群に分解する技術である．Bunch [4-2]は，クラスタ内の凝集度が高く，かつクラスタ間の結合度が低いクラスタを見つけるグラフクラスタリング手法である．SArF は共通のフィーチャーを持つクラスを同じクラスタに集めるグラフクラスタリング手法である．自然言語処理によるクラスタリング手法には，ソースコード中の識別子やコメントなどの意味情報が用いられる[5-8]．システムの実行時の挙動を理解するために，

実行トレースなどの動的情報を用いるクラスタリング手法もある[5-7].

Scanniello ら[5-12]はレイヤーを抽出するクラスタリング手法であるが，人間の判断が必要であり，後述の図 5-11(a)のような複雑なレイヤー構造を持つソフトウェアには適用できない．SArF Map はソフトウェアをフィーチャーに分解することで，複雑なレイヤー構造を解きほぐし，レイヤーを自動的に可視化することができる．これについては 5.3.3 節で述べる．

Street and Block Tree Layout (SBTL)は，過去の研究[5-6][5-10]で用いられた階層的レイアウトアルゴリズムを拡張したものである．SBTL では，ビルではなく街区をレイアウトし，街区内のビルをレイアウトするために追加のレイアウトアルゴリズムを内包する．階層型レイアウトアルゴリズムは面積当たりの情報量の低さが欠点であるが，SBTL は 1 つの街区に多くのビルを配置するため，比較的高い面積当たりの情報量を実現している．

ソフトウェアクラスタリングには，古くから可視化ツールが用いられてきた．例えば，Bunch は graphviz tool (http://www.graphviz.org/) を用いてクラスタリングの結果を評価している．しかし，大規模ソフトウェアを対象としたソフトウェアクラスタリングと，可視化を密に連携した既存研究はほとんど無い．意味に基づくソフトウェアクラスタリングと組み合わせた手法である Software Cartography [5-8]があるのみである．

5.3 ソフトウェア可視化手法 SArF Map

本節では，まず SArF Map の概要を説明し，次に，手順を述べ，次いでその基盤となるソフトウェアクラスタリングアルゴリズムについて簡単に紹介し，手順の詳細を説明する．最後に SArF Map を実行するシステムについて述べる．

5.3.1 概要

提案手法である SArF Map は，対象となるソフトウェアの全体像をフィーチャーとレイヤーの観点から地図として可視化する手法である．生成された地図を指して SArF Map と呼ぶこともある．図 5-1 は，オープンソースのデータマイニングツールである Weka 3.0 [4-37]の SArF Map の例である．以下の節ではこの例を用いて SArF Map の概念とアルゴリズムについて説明する．

SArF Map ではソフトウェアアーキテクチャを可視化するために，都市メタファーを採用した．実際のソフトウェアにおけるエンティティや概念と，SArF Map で用いたメタファーとの対応を表 5-1 に示す．

表 5-1　ソフトウェアのエンティティや概念と都市メタファーとの対応付け

エンティティ・概念	都市メタファー
クラス／ソースファイル	ビル
フィーチャー	街区
レイヤー	北から南への斜面
フィーチャー間の関係	道路とその距離

図 5-1 の SArF Map には，色分けされた 142 件のビルと 8 つの街区が描かれている．各ビルがクラスを表し，各街区は何らかのフィーチャーを持つクラスの集合を表す．街区の上に表示されている語は，その街区のフィーチャーを表すキーワードである．例えば，濃緑色のビルが並ぶ左上の街区は，“estimator”という語で特徴づけられており，そのフィーチャーが，データマイニングツールの一般的な機能の 1 つである「指標算出」である可能性があることが分かる．地図上の道路はフィーチャー間の関連性を表し，色のついた曲線はクラス間の依存性を表す．詳細な説明は後の節で述べる．

SArF Map の入力データは依存性グラフのみである．本章の結果はすべて対象ソフトウェアの JAR ファイルから抽出している．抽出された依存関係は，メソッド呼び出し，フィールドアクセス，継承，クラスタイプの参照である．クラスタリングの結果，すべてのメンバレベルの依存関係は，クラスレベルの依存関係にまとめられる．そのため，生成された地図にはクラスレベルの依存関係のみが描かれ，ユーザはメンバレベルを意識する必要はない．SArF Map の生成は完全自動であり，ユーザの介入は不要である．

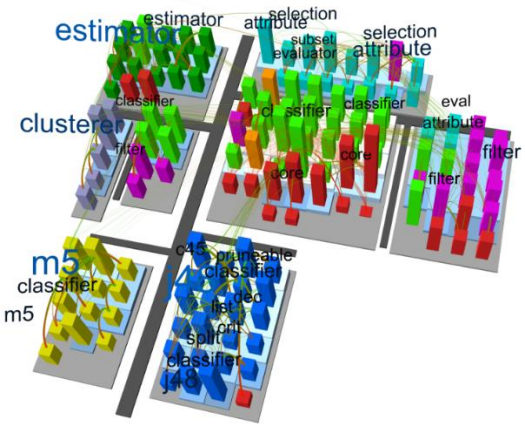


図 5-1　Weka 3.0 の SArF Map (142 クラス, 9 パッケージ)

5.3.2 手順

SArF Map の手順は図 5-2 に示す通り，以下のステップで構成されている．

(1) 4 章で述べた SArF クラスタリングアルゴリズムを用いて，対象システムに対してソフトウェアクラスタリングを実行し，フィーチャーを表すクラスタを得る．その結果，図 5-2(a)に示すようなデンドログラムが得られる．青い破線の円で囲まれた部分木がクラスタである．

(2) デンドログラム中の各クラスタ（フィーチャー）を縮退して葉ノードとし，図 5-2(b)に示すようなフィーチャーを葉ノードに持つ抽象木を生成する．デンドログラムの枝ノードは，より近いノードが同じ枝またはより近い枝に属するように展開する．

(3) 図 5-2(c)のように，フィーチャーごとに，街区内のビル（クラス）をレイアウトする．このレイアウトアルゴリズムは，各クラスが属するレイヤーに応じて位置を決定する．詳細は 5.3.3 節で述べる．

(4) 道路上にフィーチャーを配置する．木の枝を道路として表現する．関係のあるフィーチャーの街区を可能な限り近くに配置する．このステップで，図 5-2(d)に示すように，ユーザにとってのメンタルモデルとなる二次元の白地図が生成される．詳細は 5.3.4 節で述べる．

(5) ユーザが分析に用いる情報を白紙の地図に重ねる．詳細は 5.3.5 節で述べる．

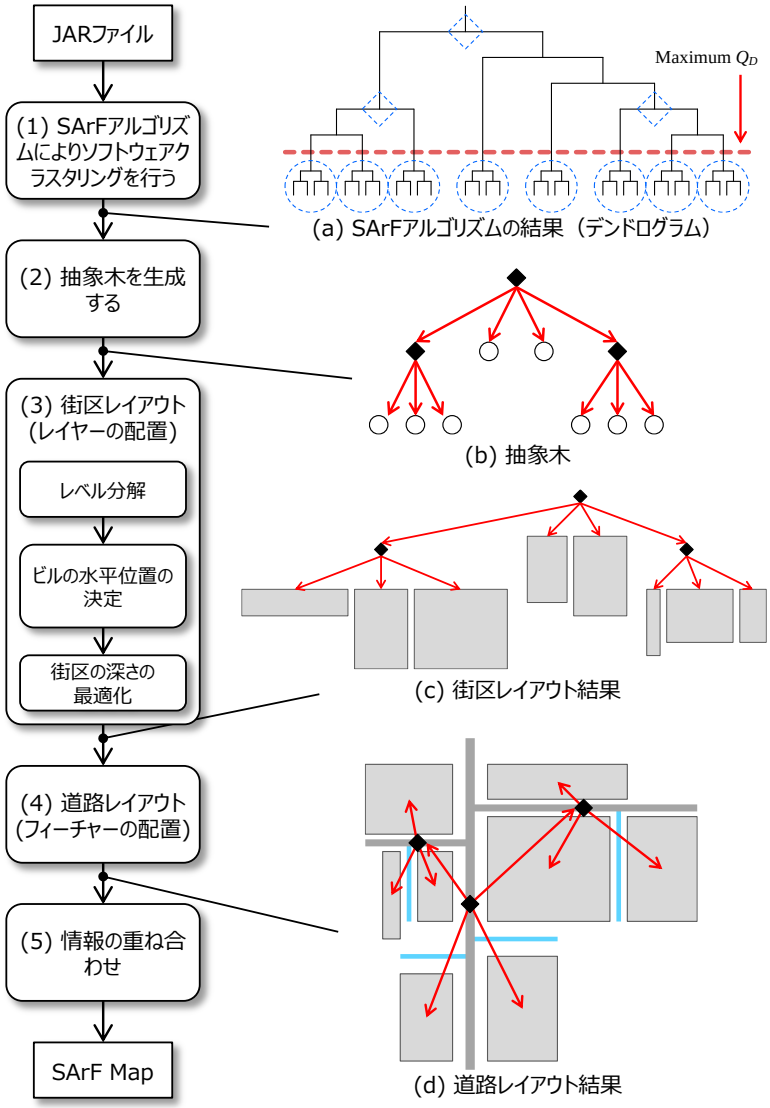


図 5-2　SArF Map の手順と処理中のデータ構造

5.3.3 レイヤーのための街区レイアウトアルゴリズム

ソフトウェアアーキテクチャにはしばしばレイヤー構造が用いられる．特に，よい設計と言われるソフトウェアは，明確に定義されたレイヤー構造を持つ傾向がある．レイヤー間の依存関係は一方向であるため，レイヤー構造を把握することで，ソフトウェアの理解が容易になる．また，レイヤー構造を持たないことや循環的な依存関係があること分かれば，ソフトウェアの保守性に悪影響を及ぼすと判断することができる．

SArF Map では，都市メタファーの下でレイヤー構造を可視化するために，街区の中に高低差（北から南に下る斜面）を導入し，上位のレイヤーに属するクラス（ビル）を高い位置（北側）に配置し，下位のレイヤーに属するクラス（ビル）を低い位置（南側）に配置する．レイアウトの手順は，図 5-3 に示すように，(1) レベル分解，(2) ビルの水平位置レイアウト，(3) 街区の深さの最適化，の 3 つのステップからなる．

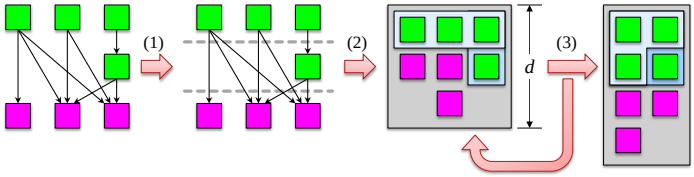


図 5-3 街区レイアウトの手順

(1) レベル分解：図 5-3 の最初のステップでは，クラスタ内のクラスの依存関係グラフをいくつかのレベルに分解する．本章でいうところのレベルとは，図 5-4 に示すように，レイヤーを細分したものであり，クラスタ内の依存関係が最上位または最下位のクラスからほぼ同じ距離にあるクラスのグループであると定義する．レイヤーの決定は自明ではないが，レベルの決定は図 5-4 のように自動的に行うことができる．

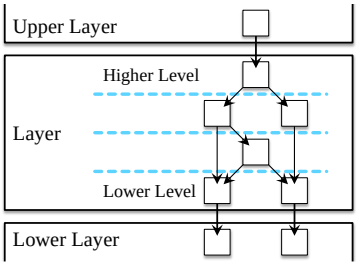


図 5-4　レイヤーとレベル

レベルとして望ましい性質は，下位のレベルにのみ依存し，上位のレベルにのみ依存されることである．しかし，実用的なソフトウェアは循環依存関係を持つことが多く，古典的なトポロジカルソートアルゴリズムではレベルの判別ができない．そこで，図 5-5 に示す貪欲レベル分解アルゴリズムを開発した．図の 11 行目で古典的な貪欲循環除去ヒューリ

スティック[5-13]を用いて循環依存関係を除いている．このアルゴリズムにより，図 5-3 に示すように，依存関係グラフを複数のレベルに分解することができる．

```
# G is the set of the input nodes.
func GreedyLevelDecomposition(G) {
  UpLevels, LowLevels := () # List of Sets
  while (|G| > 0) {
    # ins(n) returns the set of predecessor nodes of n in G
    UL := {n in G where |ins(n)| = 0} # Set
    # outs(n) returns the set of successor nodes of n in G
    LL := {n in G - UL where |outs(n)| = 0} # Set
    if (|UL| + |LL| = 0) {
      # Greedy Cycle Removal heuristic
      UL += argmax(n in G) (|outs(n)| - |ins(n)|)
    }
    for c in UL + LL {
      G -= c
      for n in outs(c) { ins(n) -= c }
      for n in ins(c) { outs(n) -= c }
    }
    if (|UL| > 0) UpLevels := UpLevels + UL
    if (|LL| > 0) LowLevels := LL + LowLevels
  }
  return UpLevels + LowLevels # List of Sets
}
```

図 5-5　貪欲レベル分解アルゴリズム

(2) ビルの水平位置の決定：図 5-3 の 2 番目のステップでは，各レベルを降順に矩形の街区に詰めていく．各ビル（クラス）は，依存関係にある隣接レベルのビルに対して，水平方向にできるだけ近い位置に配置する．そのために，以下のエネルギー関数を最小化するビルの x 座標を求める．

$$f(x_i|i) = \sum_{j\in \mathrm{ins}(i), l_j \neq l_i} \left(d_{ji}(x_i - x_j)\right)^2 + \sum_{j\in \mathrm{outs}(i), l_j \neq l_i} \left(d_{ij}(x_i - x_j)\right)^2$$

ここで，i と j はビル，x_i，l_i，x_j，l_j はそれぞれの x 座標とレベルである．i に依存するビルの集合を ins(i)，i が依存するビルの集合を outs(i)，d_{ji} と d_{ij} はそれぞれ j から i への専念度スコアと i から j への専念度スコアである．各レベルにおいて，ビルのすべての x 座標は，エネルギー関数の合計を最小化するように決定される．係数として専念度スコアを用いることで，依存関係の強さを考慮した配置がなされる．

(3) 街区の深さの最適化：図 5-3 の 3 番目のステップでは，街区の深さ（図中の「d」）を決定する．ここでは，街区の深さが依存関係の深さを表している．依存関係の理解が容易になるような最適な深さ d を求めるために，以下のペナルティ関数を最小化する d の値を，d を変化させながら 2 番目のステップを繰り返すことで探索する．

$$g(E) = \sum_{(i,j)\in E} \begin{cases} d_{ij}(|y_i - y_j| + a) & : \text{逆配置} \\ d_{ij}(b|y_i - y_j|) & : \text{順配置} \end{cases}$$

ここで，E は依存関係の集合であり，(i,j)は i が j に依存していることを表し，y_i と y_j はそれぞれ街区内の i と j の y 座標である．「逆配置」とは不適切な配置，すなわち，i が j より下位か同位になっている条件を表し，その場合にペナルティを与える．定数 a はペナルティの強さを制御するパラメータである．順配置の項は，街区の深さと幅のバランスをとるために導入され，定数 b はそのバランスを制御するパラメータである．既定値として経験的に，a は 2，b は 0.3 としている．図 5-3 の例では，左から 3 番目の街区には 1 つの逆配置があり，ペナルティを受けるが，4 番目の街区には逆配置はないため，後者の方が望ましい．

d が最小化され 3 番目のステップが完了すると，ビルの街区内の座標が確定するため，街区の都市メタファーによる可視化を行う．上位レイヤーに属するビルは北側に，下位レイヤーに属するビルは南側に配置される．さらに，必要に応じ，上下関係を視覚的に把握しやすいよう，図 5-6 のように，レベルを斜面として表現する．

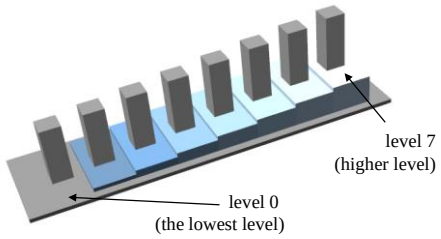


図 5-6　街区内のレベルの表現としての斜面

5.3.4 フィーチャーのための道路レイアウトアルゴリズム

この節ではフィーチャーのレイアウトの詳細について述べる．ここまでで，フィーチャーを葉にもつ抽象木とすべての街区のレイアウトが決定されているので，このステップで，すべてのクラスの 2 次元空間への配置が完了する．

このステップでの目標は，関連の強いフィーチャーを近い位置に配置することである．デンドログラムはノードの近さを表しており，抽象木はその性質を受け継いでいる．したがって，抽象木を 2 次元空間にマッピングすることで，目標が達成される．関連するフィーチャー間の距離を最小化するために，ここでは以下のエネルギー関数を最小化する．

$$h(E) = \sum_{(i,j)\in E} {d_{ij}}^2\left((x_i - x_j)^2 + (y_i - y_j)^2\right)$$

ここで，E は依存関係の集合であり，(i,j)はビル i がビル j に依存していることを示し，x_i, y_i, x_j, y_j はそれぞれ i と j の座標である．

レイアウトアルゴリズムの詳細は以下の通りである．

(1)　抽象木のルートノードを訪問する．

(2) 訪問しているノードが枝ならば，道路を配置し，その各子ノードを訪問して再帰的に配置を行う．その際には親ノードと子ノードを直交した向きとする．

(3) すべての子ノードの配置が終わったならば，各子ノードに対して，その部分木や街区を，道路に沿って，エネルギー関数 h を最小化する位置に配置する．

(4) エネルギー関数が減少している間，(1)を繰り返す．

アルゴリズム終了後に，街区の分離を視覚敵に補助するため，図 5-2(d)に示す水色の道路を追加で配置する．以上で，2 次元空間上に SArF Map の白地図が完成する．

5.3.5 2 次元白地図への情報の重ね合わせ

この節では，白地図の SArF Map の上に適用可能な情報の表現ついて述べる．3 次元空間を付加的な情報の表現のために利用できるように，SArF Map の白地図は 2 次元空間内に表現されている．白地図は複数のユーザにとって，また複数の用途にとって共通のメンタルモデルとなることを想定しており，ユーザの混乱を避けるために，どのような情報が追加されても不変であり，一貫性を保つ．

1）ソースコードの構造： パッケージや，ディレクトリ，アーキテクチャのドキュメントなどのソースコードの構造と，フィーチャーを比較することで，「ソフトウェアがどれだけよい構造となっているか？」「設計と現実のアーキテクチャの間にギャップがないか？」などの重要な問いに答えることができる．SArF Map では，ソースコードの構造に色を割り当てることで，あるフィーチャーがどのパッケージで構成されているか，また，あるフィーチャーにおいてパッケージがどのように相互作用しているかを知ることができる．図 5-7 は街区に見られる典型的な色のパターンの例である．その解釈について述べる．

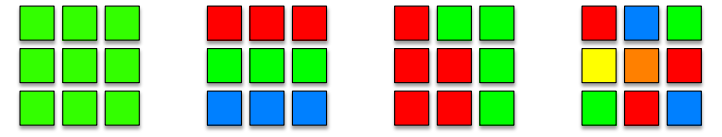
① 単色パターン(single-color)は，そのフィーチャーが 1 つのパッケージで構成されていることを表す．そのパッケージが 1 つのフィーチャーに現れるのみならば，そのパッケージとフィーチャーが一致していることを意味し，よいパッケージ構造と言える．パッケージが複数のフィーチャーに現れる場合，それらのフィーチャーが，パッケージを分割するために有効な知識となる．

② 層状パターン(layered)は，そのフィーチャーが複数のパッケージを積み重ねる形で実装されていることを表す．このパターンもよいパッケージ構造がと言える．この場合，対象ソフトウェアのパッケージ作成方針がレイヤーに基づくものであり，フィーチャーはパッケージに反映されずに暗黙のものとなる．そのため，SArF から抽出されたフィーチャーの知識は価値が大きい．

③ サブグループパターン(subgroups)は，そのフィーチャーが複数のパッケージで実装されている，または，複数のフィーチャーが共存していることを表す．これらはリファクタリングの契機となる．

④ 混合パターン(mixed-color)は，そのフィーチャーの実装が様々なパッケージに分散していることを表し，パッケージの構造が悪いか，そのフィーチャーが横断的なものであることを示唆している．

以降も，特に断りのない限り，本章のすべての SArF Map において，ビルの色はそのパッケージを表す．



(a) *Single-color* (b) *Layered* (c) *Subgroups* (d) *Mixed-color*

図 5-7　街区内のソースコード構造パターン

2）依存関係リンク： SArF Map では依存関係のリンクを理解が容易なように表示する．街区はグラフのクラスタリング結果から生成されるため，図 5-8 の例のようにほとんどの依存関係リンクは街区の中に納まるため，街区間の依存関係が際立ち，容易に追跡することができる．依存関係リンクは図に示すように曲線として表示される．曲線の向きと他の曲線との識別が容易なように，始点から終点まで，緑色から赤色まで段階的に変化させている．図ではほとんどの街区内の曲線は，北から南に向かっている（緑色が上側，赤色が下側）．街区間の依存関係をわかりやすく可視化するために，先行研究[5-10][5-11]と同様に，Hierarchical Edge Bundle[5-14]を 3D 拡張したものを利用した．各曲線の幅は，依存関係の専念度を表す．そのため，重要な依存関係が視覚的に強調される．

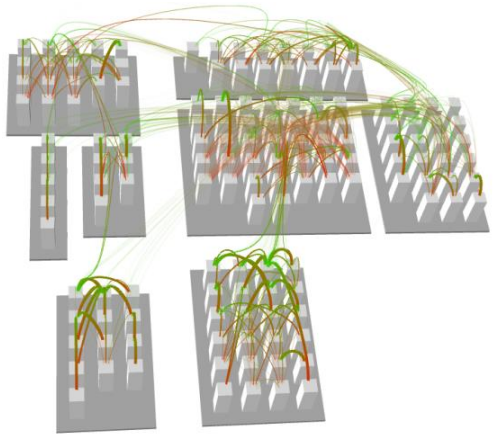


図 5-8　SArF Map の依存関係リンクの表現 (Weka 3.0)

3）キーワード： 図 5-1 に示すように，各街区を特徴づけるキーワードが街区の上に表示される．キーワードの抽出手順は以下の通りである．(1) パッケージ名やクラス名から単

語を収集する．(2) 各単語について，街区を文書とみなして，tf-idf を計算する．(3) 各単語に元のクラス（ビル）と同じ座標を与える．(4) tf-idf 値の単位面積あたりの局所密度が高い単語を選択して表示する．(5) 表示された単語の位置が重って見えにくくならないように調整する．この可視化においては，大きく表示された単語はその街区の中でより特徴的な単語であることを意味する．

4) その他の特性： SArF Map のメタファーにおける可視化特性は，ビルのグリッド上の位置を除いて，自由に利用が可能で，分析に必要なあらゆるメトリクスをマッピングすることができる．典型的な使用例を，前述の使用例を含めて表 5-2 に示す．

表 5-2　SArF Map メタファーにおける可視化特性のマッピング

メタファー	可視化特性	表現される対象
ビル	色	パッケージ，カテゴリ，リスク
	高さ	サイズ（LOC やメソッド数）
	形	ファイルタイプ
	ずれ，回転，傾き，テクスチャ	標準無し（任意のメトリクス）
ビルの装飾	明度	利用度
	火災	欠陥，問題点
地面	色	レイヤーレベル，カテゴリ
	高さ	レイヤーレベル
リンク	色	タイプ，方向
	太さ	重要度

5.3.6 支援システム

図 5-2 の手順で SArF Map を作成するツールと，作成した SArF Map を表示する 3D ビューワーの 2 つを開発した．ビューアは OpenGL (http://www.opengl.org/)を用いて実装されており，SArF Map に対して 3D のインタラクティブなナビゲーション機能，リンク解析機能，表 5-2 に示した可視化特性のマッピング機能を提供する．

5.4 リサーチクエスチョン

SArF Map の有効性を評価するために，以下のリサーチクエスチョンを設定した．ケーススタディでは，これらを評価する．

RQ1: SArF Map はフィーチャーを可視化できるか？

RQ2: SArF Map はレイヤーを可視化できるか？

RQ3: パッケージの構造品質の評価に SArF Map を使用できるか？
RQ4: アーキテクチャの知識を可視化できているか？
RQ5: SArF Map は様々なステークホルダーに一貫したビューを提供できるか？

5.5 ケーススタディ

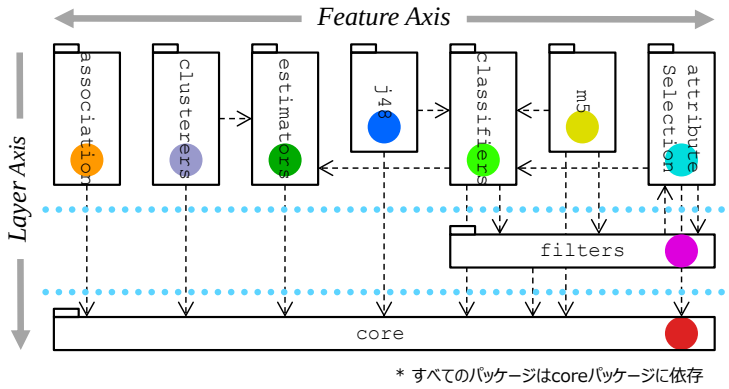
SArF Map の有効性を評価するために，オープンソースとプロプライエタリの両方のソフトウェアから 5 つのケーススタディを実施する．これらはすべて Java 言語で実装されている．SArF Map が出力したフィーチャーの正しさを評価するためには，対象ソフトウェアに詳しい知識を持つ者（オーソリティ）からフィーチャーの観点に基づく正解を得る必要がある． SArF Map が出力した結果を「算出されたフィーチャー」と呼び，オーソリティからの正解を「オーソリティ・フィーチャー」と呼ぶ．5.5.1 節から 5.5.4 節のケーススタディでは，パッケージ構造がオーソリティ・フィーチャーになっているか，またはその一部がオーソリティ・フィーチャーになっており，それらをケーススタディに先立って特定を行った．このケーススタディでは，ビルの色を識別しやすくするために，図 5-15 を除くすべての SArF Map でビルの高さを固定した．

各ケーススタディで 5.4 節に述べたリサーチクエスチョンを検証する．5.5.1 節から 5.5.4 節では RQ1 から RQ4 を，5.5.5 節では RQ5 を検証する．対象ソフトウェアによっては，問うことのできないリサーチクエスチョンもある．例えばレイヤー構造がないソフトウェアでは RQ2 は意味が無く，アーキテクチャ知識が隠れていなければ RQ4 は意味が無い．

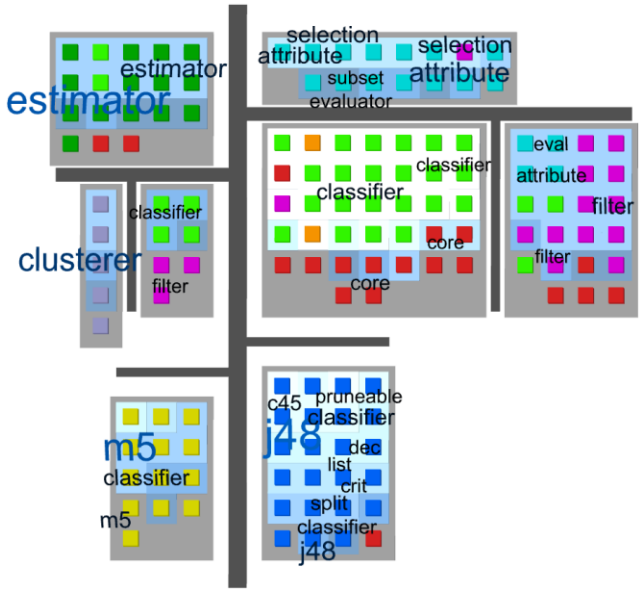
5.5.1 Weka 3.0（オープンソース）

Weka はオープンソースのデータマイニングツールである．そのバージョン 3.0 のアーキテクチャは[4-37]に詳細に説明されており，コア以外のすべてのパッケージはそのフィーチャーに対応しており[4-12]，オーソリティ・フィーチャーとみなすことができる．図 5-9(a)は，Weka のアーキテクチャを示している．このアーキテクチャは 3 つの層で構成されている．図 5-9(b)はその SArF Map である．(a)と(b)のパッケージの色の割り当ては同一である．

SArF Map を見ると，次のことが分かる．(1)ほとんどの街区が(a)のオーソリティ・フィーチャーと一致しており，SArF Map が有効に機能していると言える（RQ1 を支持）．(2)いくつかのブロックでは，層状パターンが見られ，(b)から読み取れるレイヤーは(a)のレイヤーと一致している．これは，SArF Map がレイヤーを効果的に可視化していることを意味している（RQ2 を支持）．(3)ほとんどのフィーチャーが単色パターンまたは層状パターンであるため，パッケージ構造がよいと判断することができる（RQ3 を支持）．パッケージ構造からアーキテクチャが自明なため，RQ4 は問わない．



(a)　Weka 3.0 のアーキテクチャ（パッケージ図）



(b)　Weka 3.0 の SArF Map（鳥瞰図）

図 5-9　Weka 3.0 のアーキテクチャと SArF Map

5.5.2 DMTool（プロプライエタリ）

DMTool（仮名），著者が所属する企業の商用データマイニングソフトウェアである．その開発者に直接インタビューできる環境にあるため，正確なオーソリティ・フィーチャーを採取することができた．インタビューに際し，開発者には（[4-3]で定義された）フィーチャーとそのフィーチャーを実装したクラスを特定してもらった．その詳細は[4-16]に記載されている．図 5-10(a)は，DMTool のアーキテクチャを示している．DMTool のパッケージ設計方針がレイヤーに基づくことが図から分かる．DMTool は，二つのレイヤー構造を持っており，右半分が 2 層，左半分が 4 層になっている．
図 5-10(b)と図 5-10(c)はその SArF Map である．キーワードは守秘義務のため表示していない．(b)の色の割り当ては(a)と同じだが，(c)はオーソリティ・フィーチャー毎に色を付けている．

この 2 つの SArF Map から，以下のことが分かる．

(1) (b)のいくつかの街区では層別パターンが見られ，観測されたレイヤーは(a)のレイヤーと一致している．これは，SArF Map がレイヤーを効果的に可視化していることを示す（RQ2 を支持）．

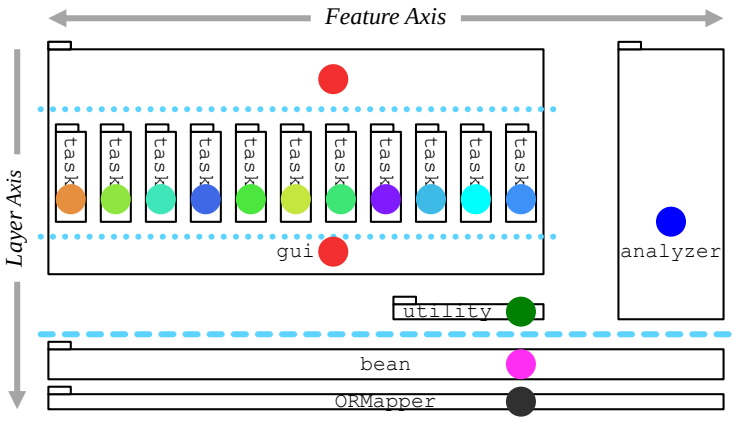
(2) (b)を見ると，赤いクラスが多くのフィーチャーに分散していることが分かる．これは GUI パッケージに含まれるフィーチャーの数が多すぎるため，分割すべきであることを示す（RQ3 を支持）．

(3) (c)では，ほとんどの街区がオーソリティ・フィーチャーと整合している．これは，SArF Map が有効に機能していることを表す（RQ1 を支持）．パッケージ構造設計がレイヤーに基づいているため，パッケージにはフィーチャーを示唆する情報が含まれていない．SArF Map はそのような暗黙の知識の復元に成功している（RQ4 を支持）．

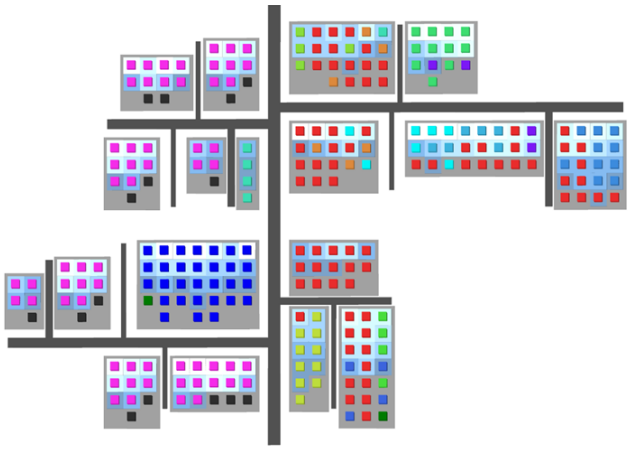
(4) (c)の左半分では，複数の隣接する街区に同じ色が跨っている．これは，SArF クラスタリングアルゴリズムが街区を過剰に分割していることを意味する．これについて開発者に確認したところ，分割された街区は開発者が作成したオーソリティ・フィーチャーの一段階細粒度のフィーチャーと一致しており，この分割は妥当である述べた．このことから，SArF Map が関連するフィーチャーを近傍に配置する性質も確かめられる．

(5) SArF Map は大まかに 2 つの部分に分けられており，その分け方はアーキテクチャと一致している（RQ4 を支持）．

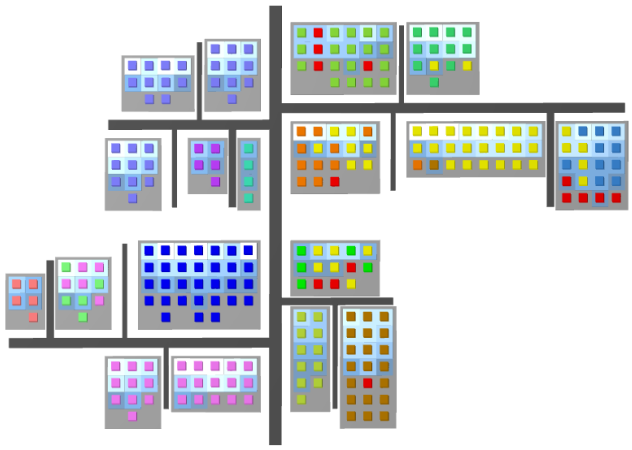
上記の観察結果を検証するため，開発者に追加のインタビューを実施した．質問は，「上記の 5 つの観察結果は有効か？」，「可視化されたソースコードの構造パターンは妥当か？」，「関連するフィーチャーがお互い近い位置に配置されているか？」であり，回答はすべて肯定であった．



(a)　DMTool のアーキテクチャ（パッケージ図）



(b) パッケージで色付けした DMTool の SArF Map（鳥瞰図）



(c) フィーチャーで色付けした DMTool の SArF Map（鳥瞰図）

図 5-10　DMTool の SArF Map（253 クラス，16 パッケージ，16 フィーチャー）

5.5.3. Javassist (オープンソース)

Javassist は Java のバイトコード操作ライブラリであり，そのパッケージ設計はフィーチャーに基づいている．図 5-11 は Javassist 3.16.1 の SArF Map である．ビルの色は，パッケージ名の辞書順に近いパッケージが似た色にあるように割り当てた．これは，次のケーススタディでも同様である．ほとんどの街区が単色パターンとなっている．これは，フィーチャーに基づくという Javassist のパッケージ設計方針が厳密に守られていることを意味している（RQ3 を支持）．また，街区とパッケージの一致率が高いことから，SArF Map はフィーチャーの収集に成功していると言える（RQ1 を支持）．レイヤー構造を持たないため RQ2 は問わず，パッケージ構造からアーキテクチャが自明なため，RQ4 は問わない．

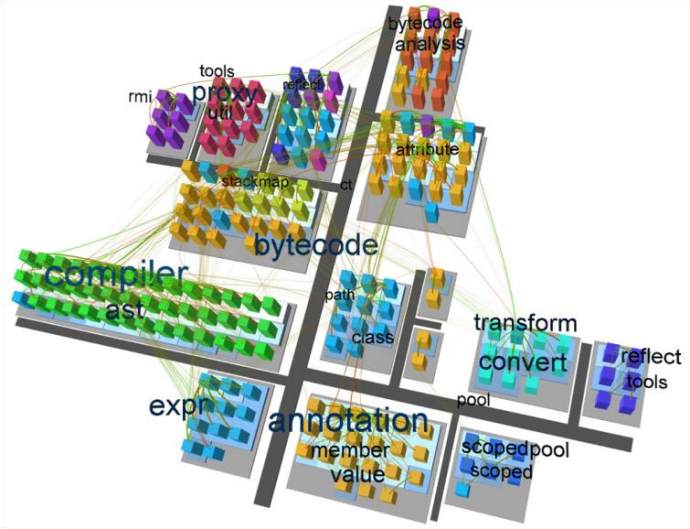


図 5-11　Javassist 3.16.1 の SArF Map（206 クラス，15 パッケージ）

5.5.4 JDK Swing（オープンソース）

JDK Swing は，Java の GUI ウィジェットツールキットで，図 5-13 はその SArF Map である．多くの街区に緑・赤・黄色・黄緑色のビルからなる層状パターンが見られる．このパターンは図 5-12 に示す通り，Swing API の典型的な依存関係とそのレイヤー構造である．SArF Map が Swing のレイヤー構造を効果的に可視化していることを示している（RQ2 を支持）．また，このことから，Swing のパッケージの設計方針がレイヤーに基づいていると判断できる．

層状パターンの街区には，“table”，“border”，“combo”など GUI として意味が明確なキーワードが表示されており，各フィーチャーの機能を容易に推測できる．これらのキーワードと街区内のクラスの実際のフィーチャーと比較し，整合していることを確認した．よっ

て，SArF Map がフィーチャーを効果的に可視化し，暗黙の情報を明らかにしたと言える（RQ1 と RQ4 を支持）．ほとんどのフィーチャーが単色パターンまたは層状パターンであるため，パッケージ構造がよいと判断することができる（RQ3 を支持）．

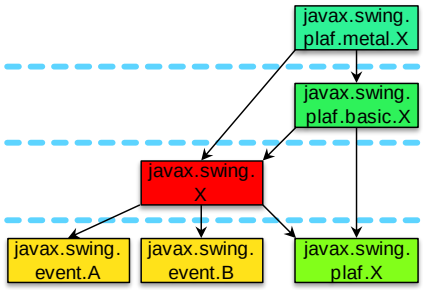


図 5-12　JDK Swing の典型的な依存関係とレイヤー構造

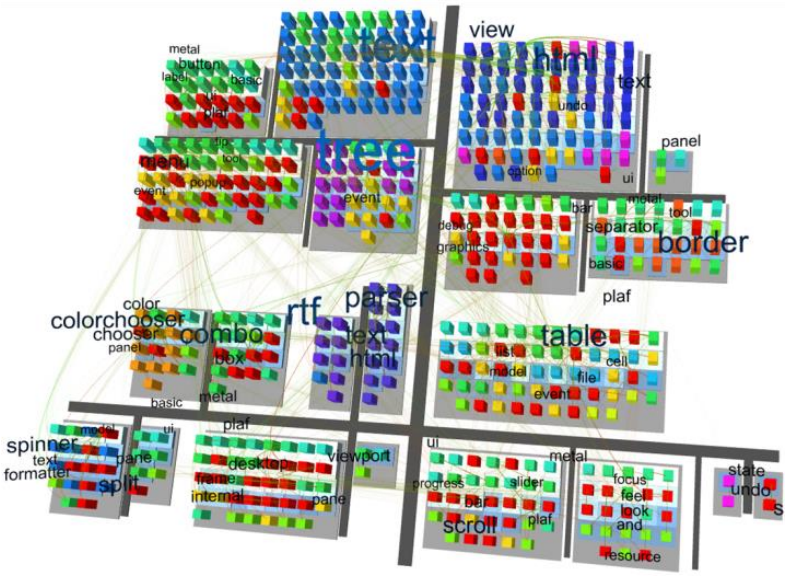


図 5-13　JDK Swing 1.4.0 の SArF Map（536 クラス，16 パッケージ）

5.5.5 従業員管理システム（プロプラエタリ）

このケーススタディでは，著者が所属する企業が開発した従業員管理システム(EMS)の 1 つを可視化する．このケーススタディの目的は，RQ5「SArF Map は様々なステークホルダーに一貫したビューを提供できるか？」を評価することである．このケーススタディでは，2 つのビューを扱う．(1)開発者ではない意思決定者のためのビューと，(2)保守を行う開発者のためのビューである．この 2 種のステークホルダーのニーズを満たすために，同じ SArF Map の白地図に，全く異なる 2 つの情報セットを重ね合わせることで 2 つのビューを用意した．

5.5.5.1 意思決定者向けビュー

図 5-14 は，高レベルな意思決定者のための SArF Map である．ここでは，保守性，利用度，最近の更新状況を，それぞれ雑然性，明度，建設中のビルとして表現している．街区はフィーチャーを意味するので，意思決定者が「建設中の明るい雑然とした街区」を観察したとしたら，それは「品質リスクの大きい利用度の高い拡大中の業務機能」と解釈することができる．利用度は，クラスやソースファイルの実行頻度のことで，実行トレースを使って測定することができる．保守性は，2 章と同様の方法で対象クラスの欠陥数を予測し，予測欠陥数の少なさを保守性の高さとした．

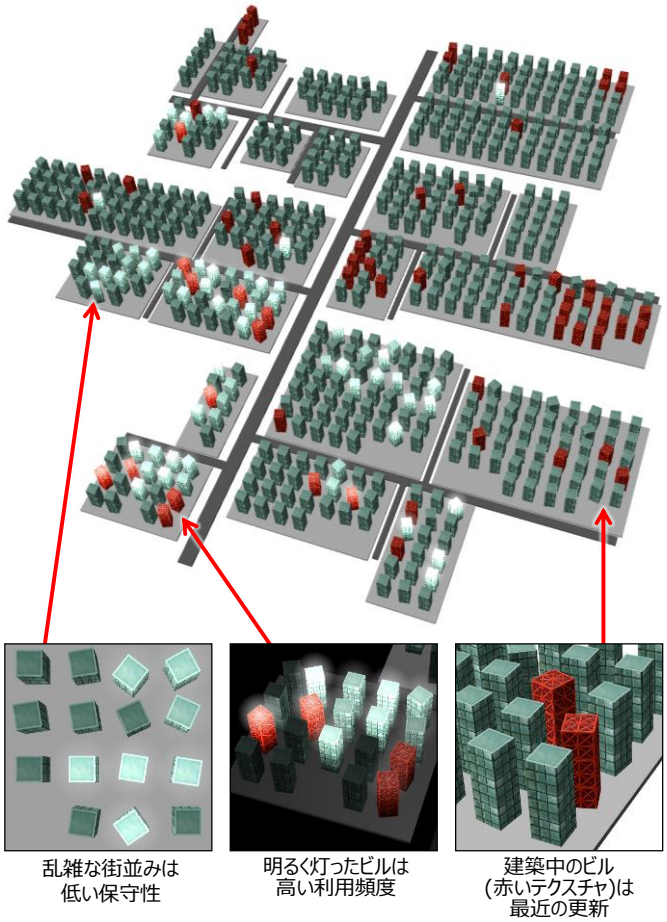


図 5-14　意思決定者向けの従業員管理システムの SArF Map（570 クラス）

5.5.5.2 保守開発者向けビュー

図 5-15 は開発者向けの SArF Map で，図 5-14 と同じ白地図を共有しており，変更の影響分析やリスク評価などの保守作業に用いる．このビューでは，ビルの高さがメソッドの数を表し（実際には，その平方根を用いる），色は 2 章で述べたインパクトスケール[2-17]を表している．インパクトスケールは変更影響量のメトリクスであり，リスク，労力，品質の評価に用いられる．インパクトスケールの値が高いと，高リスクを意味し，ここでは赤色の強さで表現している．

変更の影響を具体的に調べるために依存関係のリンクも表現する．例えば，図 5-15 の矢印で示した黄色のクラスに注目すると，そのクラスが関わるすべての依存関係のリンクが，3D ビューアツールによって強調表示される．これを用いて，開発者は変更の影響を簡単に調べることができる．

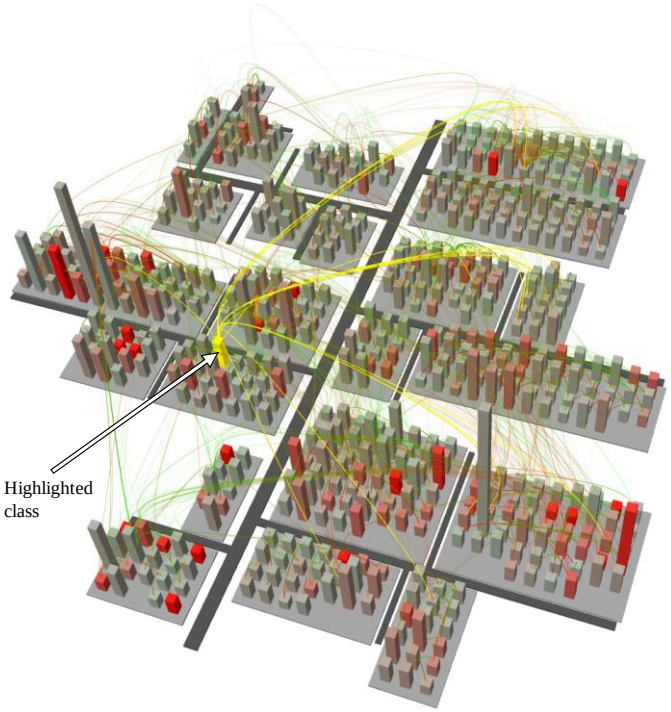


図 5-15　保守開発者向けの従業員管理システムの SArF Map（高さはメソッド数，色はインパクトスケール）

5.5.5.3 考察

2 つのビューはそれぞれ異なる情報を提供しているが，同一の白地図を共有している．そのため，高レベル意思決定者と開発者が同じメンタルモデルの上でコミュニケーションが可能になる（RQ5 を支持）．保守開発者のユースケースとしては，システム上でリスクを抱える部分の認識を共有することで予防保全のための予算獲得交渉に活かす，などがある．この SArF Map を用いたインタビューの結果，メンタルモデルを共有するための課題

として，フィーチャーによい名前を付けることが意思決定者の理解のために影響が大きいことが分かった．現在の比較的ナイーブなキーワード選択手法（tf-idf の密度）によって抽出されたキーワードは，意思決定者にとって満足できるものではなく，円滑なコミュニケーションを図るために保守開発者がキーワードを理解しやすい名前に付けなおす必要があった．

5.5.6 リサーチクエスチョンへの回答

既に述べたように，すべてのリサーチクエスチョン(RQ)はケーススタディで支持された．その上，結果を一般化し，多くの実践的状況で SArF Map による可視化がどれほど効果的であるかを実証するためには，さらなる実験と分析が必要となるだろう．

5.6 妥当性への脅威

最も重要な脅威は，4 章に述べた SArF クラスタリングアルゴリズムによってフィーチャーの収集が成功したという仮説である．4 章及び[4-16]ではいくらかの証拠を提示しており，本章でもそれらを視覚的に確認した．しかし，この仮説を広く一般的な範囲までに検証するためには，さらなるケーススタディや実験が必要である．

もう一つの脅威は，依存関係グラフの品質である．SArF Map の品質は，抽出された依存関係グラフの品質に依存する．対象となるソフトウェアでリフレクションやディペンデンシインジェクション（依存性の注入）などが多用されている場合，不完全な SArF Map が生成されてしまう．

5.2 節の関連研究で述べた都市メタファーの利点と欠点の多くを SArF Map も引き継ぐことになる．[5-15]は SArF Map を著者が所属する企業にて実システムに対し使用した際の経験報告である．これらは統制された対照実験ではないため，定量的な言及はできないが，概ね 5.2 節に述べた従来の知見通りの結果が得られている．特に得られた学びとしては，視覚的な対象に引き摺られて誤解を招き易いこと，具体的には道路を依存関係の強さではなくデータフローだと考えたユーザが多くいたことである．また，直観的理解は早いが，そこから定量評価に繋がらないと可視化の効果が薄いという意見も多くあった．

可視化の対象をより大規模なソフトウェアシステムに広げるためには，2 つのスケーラビリティの問題が生じる．1 つ目の問題は街区の粒度である．数千以上のクラスを持つシステムを可視化した場合，数百以上のクラスを持つ大きすぎる街区が現れ，理解が困難になる．2 つ目の問題は色の解像度である．人間はあまり多くの色を識別できないので，パッケージが多すぎると，色の識別に頼ったソースコードの構造パターンが分かりにくくなる．

5.7 まとめと今後の課題

5.7.1 まとめ

本章では，新しいソフトウェア可視化技術である SArF Map を提案した．SArF Map は都市メタファーを用いて，ソフトウェアアーキテクチャをフィーチャーとレイヤーの観点か

ら可視化する．SArF Map は，4 章に述べた SArF ソフトウェアクラスタリングアルゴリズムによって抽出された，通常は明示されていない情報であるソフトウェアのフィーチャーを可視化する．フィーチャーはソフトウェアシステムにおける高レベルの抽象化単位であるため，SArF Map のユーザは対象となるシステムについて高レベルの判断を下すことができると期待される．また，容易に解釈できるため，様々な立場のステークホルダーにとって共有可能な対象システムのメンタルモデルとして使用することができる．

SArF Map がフィーチャーとレイヤーの 2 つの観点を可視化するために，Street and Block Tree Layout を開発した．このレイアウトではクラスはビル，フィーチャーは街区，レイヤーは南北方向の斜面で表現される．ビルをフィーチャーの観点から理解しやすくするために，関連するフィーチャーは道路で結ばれ，近い位置に配置される．レイアウトアルゴリズムは，エネルギー最小化問題を解くことで最適なレイアウトを求める．

本研究の主な貢献は次の 2 つである．1 つ目は，依存関係に基づくソフトウェアのクラスタリングとソフトウェアの可視化を組み合わせたことである．2 つ目は，クラスタリングアルゴリズムによりフィーチャーを抽出し，複雑なレイヤー構造を解きほぐすことで，レイヤーの分解と可視化が実現できたことである．フィーチャーとレイヤーを組み合わせることにより，例えばパッケージのソースコードの構造パターンの識別が可能になるという相乗効果をもたらす．SArF Map の白地図は敢えて 2 次元空間内に留めることで，3 次元の視覚的表現をソフトウェアの様々な情報の表示に自由にマッピングできるようにしている．フィーチャーから抽出されたキーワードはタグクラウドのように表示され，ユーザの理解の助けとなっている．依存関係のうち関心の高いフィーチャー間のもののみに注目することができるが，これは SArF Map がグラフクラスタリングを採用しているからである．

SArF Map は完全に自動化されており，出力は決定的で，入力は JAR ファイルか依存関係グラフのみである．また，言語非依存である．これらの特徴により，SArF Map は産業界の様々な実践的場面で適用可能となっている．

オープンソースソフトウェアやプロプラエタリソフトウェアを使ったケーススタディでは，SArF Map がフィーチャーやレイヤーをうまく可視化し，アーキテクチャの知識を明らかすることができ，様々なステークホルダーにも利用可能であることを示した．

5.7.2 今後の課題

今後の課題としては，SArF Map から得られた知見や効果の定性的・定量的な評価をさらに実施する必要がある．そして，SArF Map から得られた知見をどのように活かすかが問われるであろう．例えば，三神ら[5-16]は大規模ネットワーク機器ソフトウェアを対象にして，インパクトスケールなどのメトリクスを SArF Map で監視する枠組みを実際に運用し，欠陥修正の優先順位付けによる効率化が果たせたことを示した．Yano ら[5-17]は SArF Map にデータアクセスフローを重ね合わせることでレガシーアプリケーションの保守の効率化に応用した．Kamimura ら[5-18]は SArF と SArF Map を用いて，モノリシックなレガシーアプリケーションからマイクロサービスの候補となるフィーチャーを抽出する試みを行っている．また，前述したスケーラビリティ，キーワード選択，依存関係グラフの

品質などの課題についても，さらに検討する必要がある．Yano ら[5-19]はキーワード抽出の品質を向上するために自然言語処理技術にアーキテクチャの慣習名を組み合わせた．誤検出や誤認識の少ない依存関係グラフを抽出するためには，自然言語解析や動的解析と静的コード解析を組み合わせることが有望であろう．

先行研究[5-6][5-8]のようにソフトウェアの進化を可視化するためには，ソフトウェアのマイナーチェンジに強いレイアウトアルゴリズムが必要である．SArF クラスタリングアルゴリズムは変更に対して十分に頑健であるが[4-16]，SArF Map が現在使用しているレイアウトアルゴリズムはそれほど頑健ではない．今後，より頑健なレイアウトアルゴリズムを開発する予定である．

6. 結言

本研究では，ソフトウェア保守の重要な活動である欠陥修正と機能追加・マイグレーションの 2 つに着目してその効率化を目的として研究を行った．特に，継続的に知識損失が起きるソフトウェア保守の環境下においても，様々な局面で適用可能なように，知識損失の影響を受けにくいソースコードから抽出可能である依存関係グラフに基づくことを前提とした．

欠陥修正を支援する目的では，まず 2 章では，欠陥予測の精度向上を目的として，依存関係グラフに基づいて，変更の影響波及量を定量化するメトリクス「インパクトスケール」を提案した．インパクトスケールの有効性検証のため，二つの大規模企業システムを対象として欠陥予測を行った結果，10%検査工数における欠陥検出数が 50%以上上昇し，有効性を示した．

次に 3 章では，対数変換を行った線形回帰モデルは，指数曲線モデルと等価であり，ソフトウェア開発データの特徴を現すのに適していることを示した．対数変換を行ってから線形回帰を行うと，過小予測するバイアスが発生してしまうため，その補正方法について述べた．ケーススタディでは，naïve な線形回帰モデルよりも指数曲線モデルがより現実をうまく表しており，予測精度も高いこと，バイアスの補正が有効であることを示した．

これらにより，欠陥修正箇所の特定や，工数の優先付けが可能になり，限られた人的資源を効率的に保守に役立てることが可能となった．

機能追加・マイグレーションを支援する目的では，まず 4 章では，依存関係グラフに基づいて，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」を提案した．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の 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 が優れることを示した．

続く 5 章では，ソフトウェアシステムのアーキテクチャの理解を容易にするために，都市メタファーを用いてフィーチャーとレイヤーの二つの観点からソフトウェアアーキテクチャを可視化する SArF Map を提案した．フィーチャーはソフトウェアの高レベルな抽象化単位であるため，生成されたビューは再利用などの高レベルな意思決定に直接利用できるほか，開発者と非開発者であるステークホルダーの間のコミュニケーションにも役立つ．オープンソースソフトウェアとプロプラエタリソフトウェアの両方に対して行った 5 つのケーススタディを通じて，SArF Map によってソフトウェアのアーキテクチャの理解と品質評価が容易にできることを示した．

これらにより，従来人手では多大な時間のかかっていた，ソフトウェアアーキテクチャという高い抽象度の情報の復元が可能となり，またそれを人間が理解しやすいメタファーを用いて可視化することで，保守期間中に知識が失われた，または現状と乖離したソフトウェアアーキテクチャの姿を理解することができ，機能追加やマイグレーションの大きな支援となる．

上記の研究により，欠陥修正と機能追加・マイグレーションというソフトウェア保守のおいて重要な位置を占める 2 つの活動について有効な提案を果たすことができたと言える．またソフトウェア保守の課題である知識損失について，ソースコードから高抽象度の知識を復元できたことは大きな意義があると考えている．本研究により，大規模ソフトウェアの保守という社会基盤を維持するうえで欠かせない活動に対し，一定の効率化を果たすことができたと考える．

今後の課題としては，依存関係グラフを今回使用したソースコードから抽出した静的なものに限らず，実行時情報から抽出する動的依存関係グラフや，自然言語処理によって，非ソースコードドキュメントから抽出する意味的依存関係などを利用することが考えられる．動的情報は網羅性のある情報取得は困難であるが，実際の利用に応じた情報が抽出できる効果が大きいと期待できる．意味的情報は解釈に曖昧性が生じる困難はあるが，非開発者のステークホルダーにとってより解釈が容易な情報を抽出できる効果が大きいと期待できる．

参考文献

[2-1] Basili, V. R., Briand, L. C., and Melo, W. L.: A validation of object-oriented design metrics as quality indicators, IEEE Trans. Software Engineering, Vol. 22, No. 10, pp.751-761 (1996).

[2-2] Briand, L. C., Wüst, J., Daly, J. W., and Porter, D. V.: Exploring the relationships between design measures and software quality in object-oriented systems, J. Systems and Software, Vol. 51, No. 3, pp.245-273 (2000).

[2-3] Fenton, N. E., and Ohlsson, N.: Quantitative analysis of faults and failures in a complex software system, IEEE Trans. Software Engineering, Vol. 26, No. 8, pp.797-814 (2000).

[2-4] Ostrand, T. J., and Weyuker, E. J.: The distribution of faults in a large industrial software system, Proc. ACM SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA), pp.55-64 (2002).

[2-5] McCabe, T. J.: A complexity measure, IEEE Trans. Software Engineering, Vol. SE-2, No. 4, pp.308-320 (1976).

[2-6] Graves, T. L., Karr, A. F., Marron, J. S., and Siy, H.: Predicting fault incidence using software change history, IEEE Trans. Software Engineering, Vol. 26, No. 7, pp.653-661 (2000).

[2-7] Nagappan, N., and Ball, T.: Use of relative code churn measures to predict system defect density, Proc. Int'l Conf. on Software Engineering (ICSE), pp.284-292 (2005).

[2-8] Gall, H., Hajek, K., and Jazayeri, M.: Detection of logical coupling based on product release history, Proc. IEEE Int'l Conf. on Software Maintenance (ICSM), pp.190-198 (1998).

[2-9] Hassan, A. E., and Holt, R. C.: Predicting change propagation in software systems, Proc. IEEE Int'l Conf. on Software Maintenance (ICSM), pp.284-293 (2004).

[2-10] Haney, F.M.: Module connection analysis – a tools for scheduling software debugging activities, Proc. Fall Joint Computer Conference, pp.173-180 (1972).

[2-11] Cataldo, M., Mockus, A., Roberts, J. A., and Herbsleb, J. D.: Software dependencies, work dependencies, and their impact on failures, IEEE Trans. Software Engineering, Vol. 36, No. 2, pp.864-878 (2009).

[2-12] Zimmermann, T., and Nagappan, N.: Predicting defects using network analysis on dependency graphs, Proc. Int'l Conf. on Software Engineering (ICSE), pp.531-540 (2008).

[2-13] Bohner, S. A., and Arnold, R. S. (Eds.): Software change impact analysis, Bohner, S. A. and Arnold, R. S.: An introduction to software change impact analysis, IEEE Computer Society Press, pp.1-26 (1996).

[2-14] Grove, D., and Chambers, C.: A framework for call graph construction algorithms, ACM Trans. Programming Languages and Systems, Vol. 23, No. 6, pp.685-746 (2001).

[2-15] Law, J., and Rothermel, G.: Whole program path-based dynamic impact analysis, Proc. Int'l Conf. on Software Engineering (ICSE), pp.308-318 (2003).

[2-16] Ren, X., Shah, F., Tip, F., Ryder, B. G., and Chesley, O.: Chianti: a tool for change impact analysis of Java programs, Proc. Conf. on Object-Oriented Prog., Syst., Lang., and App. (OOPSLA), pp.432-448 (2004).

[2-17] Kobayashi, K., Matsuo, A., Inoue, K., Hayase, Y., Kamimura, M., and Yoshino, T.: ImpactScale: Quantifying Change Impact to Predict Faults in Large Software Systems, Proc. Int'l Conf. on Software Maintenance (ICSM), pp. 43-52 (2011).

[2-18] Sharafat, A. R., and Tahvildari, L.: A probabilistic approach to predict changes in object-oriented software systems, Proc. European Conf. on Software Maintenance and Reengineering (CMSR), pp.27-38 (2007).

[2-19] Tsantalis, N., Chatzigeorgiou, A., and Stephanides, G.: Predicting the probability of change in object-oriented systems, IEEE Trans. Software Engineering, Vol. 31, No. 7, pp.601-614 (2005).

[2-20] Whaley, J., and Lam, M. S.: Cloning-based context-sensitive pointer alias analysis using binary decision diagrams, Proc. ACM SIGPLAN Conf. on Prog. Lang. Design and Impl. (PLDI), pp.131-144 (2004).

[2-21] Mende, T., and Koschke, R.: Effort-aware defect prediction models, Proc. European Conf. on Software Maintenance and Reengineering (CSMR), pp.107-116 (2010).

[2-22] Akaike, H.: A new look at the statistical model identification, IEEE Trans. Automatic Control, Vol. AC-19, No. 6, pp.716-723 (1974).

[2-23] Chatterjee, S. and Hadi, A. S.: Regression analysis by example, 4th Edition, John Wiley and Sons (2006).

[2-24] Arisholm, E. and Briand, L. C.: Predicting fault-prone components in a Java legacy system, Proc. ACM/IEEE Int'l Symp. on Empirical Software Engineering (ISESE), pp.8-17 (2006).

[2-25] Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., and Bener, A.: Defect prediction from static code features: current results, limitations, new approaches, J. Automated Software Engineering, Vol.17, pp.375-407 (2010).

[2-26] Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K., Adams, B. and Hassan, A. E.: Revisiting common bug prediction findings using effort-aware models, Proc. IEEE Int'l Conf. on Software Maintenance (ICSM), pp.1-10 (2010).

[2-27] Cameron, A. C., and Trivedi, P. K.: Regression analysis of count data, Cambridge University Press (1998).

[2-28] Khoshgoftaar, T. M., Geleyn, E., and Gao, K.: An empirical study of the impact of count models predictions on module-order models, Proc. IEEE Int'l Software Metrics Symp. (METRICS), pp.161-172 (2002).

[2-29] Tosun, A., Turhan, B. and Bener, A.: Validation of network measures as indicators of defective modules in software systems, Proc. Int'l Conf. on Predictor Models in Software Engineering (PROMISE), pp.5:1-5:9 (2009).

[2-30] Nguyen, T., Adams, B. and Hassan, A.: Studying the impact of dependency network measures on software quality, Proc. IEEE Int'l Conf. on Software Maintenance (ICSM), pp.1-10 (2010).

[2-31] Premraj, R. and Herzig, K.: Network Versus Code Metrics to Predict Defects: A Replication Study, Proc. Int'l Sympo. on Empirical Software Engineering and Measurement (ESEM), pp. 215-224 (2011).

[2-32] Borgatti, S.P., Everett, M.G. and Freeman, L.C.: UCINet for Windows: software for social network analysis, Analytic Technologies (2002).

[2-33] Hanneman, R. A. and Riddle, M.: Introduction to social network methods, University of California, Riverside (2005).

[2-34] Jolliffe, I. T.: Principal component analysis, 2ed, Springer (2002).

[2-35] Chidamber, S. R., and Kemerer, C. K.: A metrics suite for object oriented design, IEEE Trans. Software Engineering, Vol. 20, No. 6, pp.476-493 (1994).

[2-36] Geiger, R., Fluri, B., Gall, H., and Pinzger, M.: Relation of code clones and change couplings, Int'l Conf. of Fundamental Approaches to Software Engineering (FASE), LNCS 3922, Springer, pp.411-425 (2006).

[2-37] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M., and Kusumoto, S.: Ranking significance of software components based on use relations, IEEE Trans. Software Engineering, Vol. 31, No. 3, pp. 213-225 (2005).

[2-38] 早瀬康裕，松下誠，楠本真二，井上克郎，小林健一，吉野利明：影響波及解析を利用した保守作業の労力見積りに用いるメトリックスの提案，電子情報通信学会論文誌 D，Vol.J90-D，No.10，pp.2736-2745 (2007).

[2-39] 野中誠，山田弘隆，中嶋久彰，伊藤雅子：インパクトスケールを用いた不具合修正にかかわるソフトウェア変更の予測, 情報処理学会研究報告, Vol.2018-SE-198, No.33 (2018).

[2-40] 野中誠，中嶋久彰，伊藤雅子，山田弘隆：ランダムフォレストを用いたソフトウェア不具合修正予測におけるインパクトスケールの有用性，情報処理学会研究報告，Vol.2018-SE-200，No.7 (2018).

[3-1] Boehm, B. W.: Software Engineering Economics, Prentice-Hall (1981).

[3-2] Desharnais, J. M.: Analyse Statistique de la Productivité des Projets de Développement en Informatique à Partir de la Technique des Points de Fonction, in Program de maîtrise en informatique de gestion, Université du Québec à Montréal (1988).

[3-3] Finney, D. J.: On the Distribution of a Variate Whose Logarithm is Normally Distributed, Journal of the Royal Statistical Society of London, Series B, No.7, pp.155-161 (1941).

[3-4] Foss, T., Stensrud, E., Kitchenham, B. and Myrtveit, I.: A Simulation Study of the Model Evaluation Criterion MMRE, IEEE Trans. on Software Engineering Vol.29, No.11 (2003), pp.985-995.

[3-5] Grafen, A., Hails, R.(著), 野間口謙太郎, 野間口眞太郎(訳)：一般線形モデルによる生物科学のための現代統計学, 共立出版 (2007).

[3-6] Kitchenham, B. and Mendes, E.: Why Comparative Effort Prediction Studies May Be Invalid, Proc. Int'l Conf. on Predictor Models in Software Engineering (PROMISE), Article No.4 (2009).

[3-7] Newman, M. C.: Regression Analysis of Log-transformed Data: Statistical Bias and Its Correction, Environmental Toxicology and Chemistry, Vol.12, No. 6, pp.1129-1133 (1993).

[3-8] 奥野忠一, 久米均, 芳賀敏郎, 吉澤正：多変量解析法(改訂版), 日科技連 (1981).

[3-9] Sprugel, D. G.: Correcting for Bias in Log-transformed Allometric Equations, Ecology, Vol. 64, No. 1, pp. 209-210 (1983).

[3-10] 東京大学教養学部統計学教室編：自然科学の統計学, 東京大学出版会 (1992).

[3-11] 山田茂, 高橋宗雄：ソフトウェアマネジメントモデル入門, 共立出版, pp.36-50 (1993).

[4-1] 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).

[4-2] 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).

[4-3] 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).

[4-4] 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).

[4-5] 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).

[4-6] 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).

[4-7] 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).

[4-8] 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).

[4-9] 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).

[4-10] P. Andritsos and V. Tzerpos: Information-theoretic software clustering, IEEE Trans. on Softw. Eng., vol. 31, no. 2, pp. 150-165 (2005).

[4-11] O. Maqbool and H. Babri: Hierarchical clustering for software architecture recovery, IEEE Trans. on Softw. Eng., vol. 33, no. 11, pp. 759-780 (2007).

[4-12] 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).

[4-13] Z. Wen, and V. Tzerpos: Software clustering based on omnipresent object detection, Proc. Int'l Workshop on Program Compre., IWPC, pp. 269-278 (2005).

[4-14] 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).

[4-15] M. Newman: Fast algorithm for detecting community structure in networks, Physical Review E, vol. 69, no. 6, pp. 1-5 (2004).

[4-16] 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).

[4-17] A. Clauset, M. Newman, and C. Moore: Finding community structure in very large networks, Physical Review E, vol. 70, no. 6 (2004).

[4-18] 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).

[4-19] 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).

[4-20] 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).

[4-21] 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).

[4-22] 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).

[4-23] 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).

[4-24] 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).

[4-25] 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).

[4-26] V. Tzerpos and R. C. Holt: MoJo: A distance metric for software clusterings, Proc. Working Conf. on Reverse Eng., WCRE, pp. 187-193 (1999).

[4-27] Z. Wen, and V. Tzerpos: An effectiveness measure for software clustering algorithms, Proc. Int'l Workshop on Prog. Compre., IWPC, pp. 194-203 (2004).

[4-28] M. Shtern and V. Tzerpos: Lossless comparison of nested software decompositions, Proc. Working Conf. on Rev. Eng., WCRE, pp. 249-258 (2007).

[4-29] M. Shtern and V. Tzerpos: On the comparability of software clustering algorithms, Proc. Int'l Conf. on Program Compre., ICPC, pp. 64-67 (2010).

[4-30] M. Shtern and V. Tzerpos: Refining clustering evaluation using structure indicators, Proc. Int'l Conf. on Softw. Maint., ICSM, pp. 297-305 (2009).

[4-31] 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).

[4-32] M. Newman and M. Girvan: Finding and evaluating community structure in networks, Physical Review E, vol. 69, no. 2 (2004).

[4-33] E. A. Leicht and M. E. J. Newman: Community structure in directed networks, Physical Review Letters, vol. 100, no. 11, p. 118703, (2008).

[4-34] 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).

[4-35] A. Lancichinetti and S. Fortunato: Community detection algorithms: A comparative analysis, Physical Review E, vol. 80, no. 5 (2009).

[4-36] 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).

[4-37] H. Witten and E. Frank: Data Mining Practical machine learning tools and techniques, Morgan Kaufmann (2005).

[4-38] S. Ducasse, T. Girba, and A. Kuhn: Distribution map, Proc. Int'l Conf. on Softw. Maint., ICSM, pp. 203-212 (2006).

[5-1] A. R. Teyseyre and M. R. Campo: An overview of 3D software visualization, IEEE Trans. on Vis. Comput. Graph., vol. 15, no. 1, pp. 87–105 (2009).

[5-2] P. Caserta and O. Zendra: Visualization of the static aspects of software: A survey, IEEE Trans. on Vis. Comput. Graph., vol. 17, no. 7, pp. 913-933 (2011).

[5-3] G. Langelier, H. Sahraoui, and P. Poulin: Visualization-based analysis of quality for large-scale software systems, Int'l Conf. on Automated Softw. Eng., ASE, p. 214-223 (2005).

[5-4] R. Wettel and M. Lanza: Visualizing software systems as cities, Int'l Workshop on Visualizing Software for Understanding and Analysis, VISSOFT, pp. 92–99 (2007).

[5-5] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, and R. Vuduc: Communicating software architecture using a unified single-view visualization, Int'l Conf. on Engineering Complex Computer Systems, ICECCS, pp. 217-228 (2007).

[5-6] F. Steinbrückner and C. Lewerentz: Representing development history in software cities, ACM Sympo. on Software Visualization, SoftVis, pp. 193-202. (2010).

[5-7] C. Knight and M. Munro: Virtual but visible software, Int'l Conf. on Information Visualization, InfoVis, pp. 198-205 (2000).

[5-8] A. Kuhn, P. Loretan, and O. Nierstrasz: Consistent layout for thematic software maps, Working Conf. on Rev. Eng., WCRE, pp. 209–218 (2008).

[5-9] J. I. Maletic, A. Marcus, G. Dunlap, and J. Leigh: Visualizing object-oriented software in virtual reality, Int'l Workshop on Prog. Compre., IWPC, pp. 26–35 (2001).

[5-10] P. Caserta, O. Zendra, and D. Bodenes: 3D hierarchical edge bundles to visualize relations in a software city metaphor, Int'l Workshop on Visualizing Software for Understanding and Analysis, VISSOFT, pp. 1-8 (2011).

[5-11] R. Wettel: Software systems as cities, PhD Thesis, University of Lugano (2010).

[5-12] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico: Architectural layer recovery for software system understanding and evolution, Software: Practice and Experience, vol. 40, no. 10, pp. 897-916 (2010).

[5-13] I. G. Tollis, G. Di Battista, P. Eades, and R. Tamassia: Graph drawing: Algorithms for the visualization of graphs, Prentice Hall (1998).

[5-14] D. Holten: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data, IEEE Trans. on Vis. Comput. Graph., vol. 12, no. 5, pp. 741-748 (2006).

[5-15] 小林健一：メタファーを用いた高抽象度ソフトウェア可視化実践の予備報告，情報処理学会ソフトウェア工学研究会 ウィンターワークショップ 2014・イン・大洗 (2014).

[5-16] 三神郷子，中嶋久彰：メトリクスを用いてネットワークソフトウェアの内部品質を可視化する技術，電子情報通信学会論文誌 B，Vol.J100-B，No.5，pp.356-364 (2017).

[5-17] 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).

[5-18] 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).

[5-19] K. Yano and A. Matsuo: Labeling Feature-oriented Software Clusters for Software Visualization Application, Proc. Asia-Pacific Software Engineering Conference, APSEC, pp. 354-361 (2015).