チュートリアル

『ソフトウェア工学の実証的アプローチ』シリーズ第 5 回

ソフトウェアリポジトリマイニング

門田 暁人　伊原 彰紀　松本 健一

ソフトウェアリポジトリマイニング (MSR) は，研究テーマの宝庫であり，大変魅力のある研究分野である.「ソフトウェアリポジトリ」という鉱山から，マイニング (宝探し) をすることで，新しい研究目的を見つけ出すことが研究の醍醐味となっている．特に，オープンソースソフトウェア (OSS) のリポジトリを対象とした研究が盛んであり，OSS 開発の広がりとともに MSR の分野は発展してきた．本稿では，MSR 研究の魅力，主な研究テーマ，マイニングの具体例，MSR 分野の今後の発展について述べる．

The Mining Software Repositories (MSR) is an attractive research field that has tons of research topics. It is great fun for a researcher to mine new research goals from software repositories. The MSR field has been grown up with OSS communities, as researchers have mined repositories of various open source software (OSS) projects until now. In this paper we introduce fun of the MSR field as well as its typical research topics, example of mining and future directions.

1　はじめに

ソフトウェアリポジトリマイニング (英語では Mining Software Repositories, MSR) は，近年，ソフトウェア工学において最も注目されている分野の 1 つである．その名のとおりの MSR という国際会議が 2004 年より開催され，投稿者，参加者は年々増加の一途をたどっている．MSR2012 のフルペーパー採択率は 28%であり，今やかなりの難関となっている．また，MSR を主な研究テーマとする若手研究者 — Ahmed E. Hassan, Tao Xie, Sunghun Kim, Tien N. Nguyen, Thomas Zimmerman などは，ICSE，FSE，ASE などソフトウェア工学のトップカンファレンスに次々と論文を発表している．MSR の研究者は，今や，ソフトウェア工学分野における一大勢力となった．

それでは，MSR とはいかなる分野で，なぜ注目を集めているのであろうか．これまでに，MSR に関するサーベイ論文はいくつか発表されており [15] [28] [29] [43] [50]，リポジトリの種類やマイニング技術については詳しく解説されている．そこで本稿では，技術的な解説を網羅的に行うのではなく，MSR という分野の「魅力」に焦点を当て，なぜ多くの研究者が注目するのかを解説するとともに，MSR を始めようとする研究者のためにマイニングの具体例を紹介する．また，MSR 研究の現在の研究動向と，将来の展望を述べる．

2　MSR とその魅力

2.1　MSR とは

ソフトウェアリポジトリマイニングは,「ソフトウェアリポジトリ」という鉱山から，マイニング (宝探し) をするという研究分野である．ソフトウェアリポジトリとは，狭義には，ソフトウェア構成管理システム (CVS, Subversion，Git など)，バグ管理システム (Bugzilla，Trac, Redmine など)，メーリングリストアーカイブの 3 つのデータソースを主に指す．これら

Mining Software Repositories.

Akito Monden, Akinori Ihara, Kenichi Matsumoto, 奈良先端科学技術大学院大学情報科学研究科, Graduate School of Information Science, Nara Institute of Science and Technology.

コンピュータソフトウェア, Vol.30, No.2 (2013), pp.52–65.
[解説論文] 2012 年 10 月 15 日受付.

3 つは，多くのオープンソースソフトウェア (OSS) 開発において蓄積・公開されており，研究の題材として利用可能である．そのため，OSS 開発の広がりとともに MSR の分野は発展してきた．

また，従来からの研究分野である，ソフトウェアメトリクス，プログラムの静的/動的解析といったソフトウェア自体を対象とする分野においても，構成管理システムの履歴情報や，バグ管理システムの情報とひも付けた研究が行えるようになり，さらなる発展を遂げている．

一方，広義の「ソフトウェアリポジトリ」は，ソフトウェアタグ規格 [60] [61] にあるような，多種多様なソフトウェア開発データ (プロジェクト情報，開発工数データ，進捗管理データ，テスト管理データなど) を含む．これはむしろ従来のソフトウェア工学で用いられてきたデータソースであり，大学の研究者は，企業との共同研究などを通してデータを入手し，研究を進めてきた．これらの研究においても，今日では MSR 分野の研究成果が活かされ，開発工数の削減や品質向上に応用されている [19] [34].

MSR の研究は，2 つの側面がある．1 つは「何を目的にマイニングするか」であり，もう 1 つは「どのようにマイニングするか」である．特に，前者について，新しい研究目的を見つけ出すことが MSR 研究の醍醐味となっており，従来研究と比較して，研究目的が飛躍的に広がった．従来のソフトウェア工学では，企業におけるソフトウェア開発支援が目的であり，よくある研究テーマとしては，開発工数の予測，生産性の要因分析，品質評価・予測などであった．一方，MSR では，主なデータソースは OSS 開発プロジェクトであることから，OSS 開発者や OSS 利用者の支援を目的としている．OSS 開発では，分散環境における開発者がそれぞれ多様な役割を持って協調しながら開発を行うという事情があることから，研究目的も多種多様であり，例えば,「構成管理システムにソースコードをコミットする権限を持つ「コミッター」を新たに採用したいので，その候補者となるような人材を開発者の中から見つけ出す」というような目的や,「バグ管理システムに登録されたソフトウェア故障の情報から，故障の原因となるバグが含まれているソースコードを推定する」といった目的など，非常にバラエティに富んでいる．

一方,「どのようにマイニングするか」については，リポジトリ自体が非常に詳細なプロセス・プロダクトデータ (ただしノイズも多い) を含んでいることから，様々な種類のデータを取り出すことができ，マイニングの技術的な自由度が高い点が大きな魅力である．クラスタリングや機械学習といった従来のデータマイニング技術に加えて，近年では，協調フィルタリング，シーケンシャルパターンマイニング，テキストマイニングといった他分野における新しい技術を取り入れることが盛んに行われている．また，機械学習においても，ランダムフォレストやサポートベクターマシンといった新しい方法が登場したり，テキストマイニング・クラスタリングにおける潜在的ディリクレ配分法 (Latent Dirichlet Allocation; LDA) などが登場し，新たな展開を見せている．

2.2 MSR 研究の魅力

MSR 分野の今日の発展の背景には，多くの OSS 開発データが利用できるようになったことがあるが，その根底には，OSS 開発がソフトウェア業界において重要な位置を占めるようになったことがある．従来，OSS のデータを使った研究は,「実際の (企業における) 開発現場では使えない」という批判を受けていた．しかし，Linux や Android が開発プラットフォームとして一般的になるなど，今や多くの企業では OSS を利用した開発を行っており，OSS の開発/利用支援は企業にとって必要なものとなっている．

独立行政法人情報処理推進機構による OSS 活用ビジネスの実態調査 [9] によると，日本国内のソフトウェア・情報サービス・インターネット関連の 916 社へのアンケート調査の結果，51.6%の企業は顧客向けシステムに OSS を利用しており，自社利用も含めると 66.8%となっている実態が明らかとなった．一方で，同調査では，OSS 利用時の多数の課題 (例えば，緊急時の技術的サポートを得にくい，バグの改修や顧客からの要請対応に手間がかかる，ライセンスが複雑で把握しにくい，本当に利用価値のある OSS が分かりにくいなど) を挙げているが，これらの課題に MSR 研

究の成果の活用が期待される [48].

また，企業においても構成管理システムやバグ管理システムが使われていることから，MSR 分野の成果 (例えば，リポジトリメトリクスを用いたバグ予測 [16] [17] など) は企業にとっても魅力的なものとなっている．近年では，Microsoft Research の Thomas Zimmermann，Nachiappan Nagappan らによって，Microsoft 製品の開発データを用いた MSR 研究が多数発表されている [72].

一方，研究者にとっては，次章で述べるように，OSS 開発のデータ自体が大変面白く，研究テーマが多彩で，研究しがいがある．また，それだけでなく，研究業績を上げるという面でも魅力的である．近年，MSR 分野の研究者が急増した結果，互いに論文を引用し合うことから，この分野の論文の被引用回数は多くなっている．国際会議 MSR が出来てから 3 年目にあたる MSR2006(フルペーパー採択率 48%以上) に着目し，ソフトウェア信頼性の伝統的な会議である ISSRE2006(フルペーパー採択率 37%)，ソフトウェア設計・実装・解析の自動化に関するトップカンファレスである ASE2006(フルペーパー採択率 18%)，および，ソフトウェア工学のトップカンファレンスである ICSE2006(Research Track ペーパー採択率 9%) との被引用回数の比較を行った結果，Google Scholar によると，2013 年 2 月 1 日現在において，MSR2006 ではフルペーパー (16 編) の被引用回数は，平均 45.7 回であった．一方，ISSRE2006 のフルペーパー (38 編) は 19.5 回，ASE2006 のフルペーパー (22 編) では 44.3 回，ICSE2006(Research Track の 36 編) では 68.3 回であった．このことから，ICSE2006 には及ばないが，ASE2006 とそん色がなく，ISSRE2006 を大きく上回っていることが分かった．また，ICSE2006 でもっとも被引用回数が多かった論文 (308 回) は Anvik らによる “Who should fix this bug?” [1] であり，これは MSR 分野の論文である．この結果は 2006 年のみを対象としており，また，他の会議や他分野との比較を行っていない点に留意する必要があるが，世の中にインパクトを与える論文を発表するという観点からも，MSR 分野は大きな魅力を持っている．

3 MSR の研究テーマ

表 1 は，MSR 分野における代表的な研究テーマを，ソフトウェア品質，開発プロセス，協調作業，工数，OSS 利用者，ツールに分類したものである．以降，各分類におけるテーマを紹介する．

3.1 ソフトウェア品質

ソフトウェア品質は，従来のソフトウェア工学における伝統的な研究テーマであるが，OSS 開発の広がりとともに研究のためのデータの入手が容易になった

表 1 MSR 分野の代表的な研究テーマ

ソフトウェア品質
バグモジュール予測 [6] [8] [16] [17] [18] [39] [72]
Bug localization(バグ位置の推定) [7] [56] [70]
開発者の誤りのパターン検出 [36]
バグ修正時間の予測 [63]
バグ修正誤りの分析 [57]
開発プロセスの分析
変更の分析 [12] [20] [55]，予測 [2] [3]
同時変更コードの分析 [10] [69] [71]
Patch(パッチ) 投稿・受理プロセス [53]
バグ修正プロセス [24] [49]
コードクローン分析 [35]
協調作業の分析
エキスパート推薦 [30] [56]
コミッター推薦 [22]
コミュニケーション分析 [23] [42]
Openness(開放性) の分析 [54] [68]
OSS を利用する開発者の支援
ライブラリの使用状況の分析 [44]
ソフトウェア部品検索 [21] [27] [33] [59] [66]
ソフトウェアの分類 [31] [62]
ライセンスの分析 [37]
コーディングパターン検出 [25] [64] [67]
MSR 研究者の支援
リポジトリ間のリンク付け [13] [32] [58] [65]
名寄せ [4]
MSR のサービス化 [40]

ことから，MSR 分野において盛んに研究されている．

[バグモジュール予測] 伝統的なソフトウェア工学の研究テーマの 1 つであり，ソフトウェアを構成する多数のモジュールの中から，潜在バグを含むモジュールを判別することが目的である [18]．モジュールのバグ数やバグ密度を予測する研究もこのテーマに含まれる．予測結果を利用して，バグのありそうなモジュールを重点的にレビュー，テストすることでより少ないリソースでより多くのバグを検出することを目指している．MSR 分野ならではのテーマとしては，多数の OSS を用いたベンチマーキング [8]，構成管理システムのコミットログの情報 (リポジトリメトリクス) を用いた予測 [16] [17]，従来のメトリクスとリポジトリメトリクスの比較 [6]，多数の OSS プロジェクトにまたがる予測 [72], 開発者メトリクスを用いた予測 [39] などがある．

[Bug localization (バグ位置の推定)] この研究テーマは，近年のテキストマイニング技術の発展とともに盛んに研究されるようになっており，いま最もホットなテーマの 1 つである．バグ管理システムに登録されたバグ報告 (テキストデータ) を手がかりとして，ソースコード中のどの位置にバグが潜在しているかを推定する研究である [70]．バグ管理システムには，故障が発生する状況や故障の症状が記載されているが，その原因となるバグ (欠陥) がソースコード中のどこにあるかは一般に自明ではない．従来は人手によって故障の再現とソースコード読解などを行い，バグ位置を探し当てる必要があったが，もしバグ位置を自動的に絞り込めるのであればデバッグ時間を飛躍的に短縮できることになる．この研究の発展として，バグ位置と，バグを修正すべき開発者とを結びつける研究も行われている [56]．また，類似するテーマとして，変更要求とソースコードを結び付けつける研究もおこなわれている [7]．なお，bug (fault) localization という用語は，テスト実行の結果に基づいた統計的デバッギングなど，他のテーマに対しても用いられるため，注意されたい．

[エラーパターンの検出] リポジトリに含まれる情報から，コーディングパターン，および，それに違反するエラーパターンを検出し，バグ発見に役立てる研究テーマである [36]．まず，ソースコードの変更履歴とプログラムの実行履歴から，API やメソッドの呼び出し系列のパターンを検出する．そして，各パターンとその違反パターンの出現回数をもとに，パターンの分類 (正しいパターン，エラーパターン，いずれでもないパターン) を行う．プログラミングにおけるパターンを検出するという研究は従来からも存在するが，ソースコードの変更履歴を利用できるようになったことで，研究の幅が広がった．

[バグ修正時間の予測] このテーマでは，報告されたバグが修正されるかどうかや，修正に要する時間 (日数) の予測を行う [63]．一般に，大規模 OSS 開発では，日々莫大な数のバグ報告がなされるが，そのうち修正されるのは一部である (例えば Mozilla Firefox プロジェクトでは，報告されたバグのうち 84.4%は修正されない)．従って，バグが修正されるか否か，および，修正されるとしたらいつまでに修正されるのかを知ることは，OSS 利用者にとって重要となる．

[バグ修正誤りの分析] 修正されたバグの一部は，修正が不完全であったり，修正部分以外の個所で不具合が発生するといった，いわゆるデグレードやエンバグを起こすことがある．本テーマは，そのような再修正が必要となったバグ (re-opened bug) に関する分析である [57]．OSS 開発では日々多数のバグが報告，修正されていくが，修正されないままとなるものや，修正に非常に時間がかかるもの，何度も再修正されるものなど，多種多様であり，それらについての分析は MSR 分野の 1 つのテーマとなっている．

3.2 開発プロセスの分析

OSS を利用しようとする企業にとって，どのように開発されてきたか分からないものを採用するのはリスクが伴うため，開発プロセスの実態を知ることは大変重要である．OSS 開発のプロセスは，一般的なウォーターフォール開発と異なり，既存のソースコードに対する変更 (追加，削除) の系列である．そのため，変更のプロセスの分析が行われてきた．

OSS 開発におけるソースコード変更プロセスを図 1 に示す．バグ修正要求や機能拡張要求に応えるために，OSS 開発者は他の開発者と議論を行いながらソース

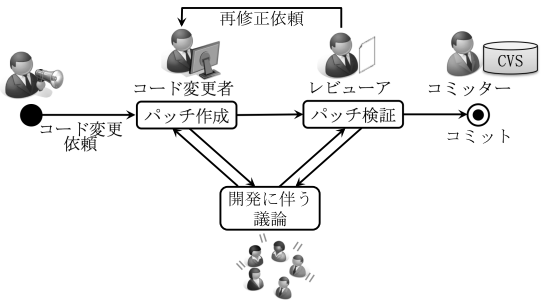

図 1　OSS 開発におけるソースコード変更プロセス

コードを変更する．その後，レビューアに patch(パッチ; 変更点 (差分情報) のみを抜き出して列挙したファイル) を提出する．レビューアも他の開発者と共にレビューを行い，問題なければ構成管理システムにコミットされプロダクトに反映される．しかしながら，大規模 OSS プロジェクトには日々数百件の要求が寄せられるため，全ての要求に応えられるわけではなく，その 10〜20%の要求しか対応されていない．このような背景を理由に，OSS 開発ならではの patch 投稿・受理のプロセスの分析や，バグ修正プロセスの分析は重要なテーマとなっている．

[変更の分析]　ソフトウェア開発はソースコードの変更 (追加，削除) の繰り返しのプロセスであり，その情報は構成管理システムに記録されている．この研究テーマでは，開発プロセスを理解するために，変更がどのように行われているか，どういった種類の変更があるか，バグとの関係，変更の粒度などの分析を行っている [20] [55]．また，行単位の変更の分析のみならず，クラス・メソッドレベルでの分析や，AST 上での分析なども行われている [12]．一般に，変更はバグ混入との関係が強い [14] ことから，本テーマはソフトウェア品質の評価にも役立つ．

[変更の予測]　このテーマでは，将来の開発におけるソースコードの変更を予測する．例えば，将来的に大きな変更が起こりそうなモジュール [2] や，変更回数が大きくなりそうなモジュール [3] は，開発やデバッグにコストがかかることから，予め把握しておくことが重要となる．

[同時変更コード (ロジカルカップリング) の分析] このテーマでは，同時に変更されることの多いソースファイル群を特定し，開発時の同時変更のし忘れを警告したり，リファクタリングの対象とすることを目的としている [10] [69] [71]．同時に変更されるファイル間の関係は，論理的な結合 (ロジカルカップリング) の一種とみなされる．

[Patch (パッチ) 投稿・受理プロセスの分析]　OSS 開発においては，patch 投稿という形で多数の人間が開発に携わることができる．しかし，大規模 OSS においては，投稿される patch の数が膨大であることから，全ての patch が採用されるわけではない．本テーマでは，どのような patch が採用されているのかを分析している [53]．

[バグ修正プロセスの分析]　OSS 開発におけるバグ修正のプロセスは，バグ報告，バグ承認，修正担当者決定，バグ修正，修正確認，といった作業を含み，さらには，バグの再修正が必要となる場合がある．このテーマでは，バグ修正プロセスの実態を可視化してボトルネックとなっている作業を発見したり，修正が早いバグと遅いバグのプロセスの違いを明らかにすることを目的としている [24]．さらに，バグ管理システムと構成管理システムの両方の履歴を分析することで，バグを修正しているにも関わらずバグ管理システムへの登録を忘れているケースや，逆に，修正したコードを構成管理システムに登録し忘れているケースを発見することも研究されている [49]．

[コードクローン分析]　開発プロセスを分析する 1 つの切り口として，コードクローンがある．Livieri ら [35] は，Linux のバージョンの遷移とクローン含有率との関係を分析している．なお，コードクローン分析技術は，本研究テーマに限らず，OSS を対象とした様々なテーマに応用されている．例えば，流用コードの発見 [47] や patch を当てるべきコードの発見 [26] などである．

3.3　協調作業の分析

OSS 開発では，互いに見知らぬ多数の開発者が協調して作業を行う必要があることから，協調作業の理解や支援を行うことが大きな研究テーマとなる [51]．

[エキスパート推薦]　OSS 開発は非常に多くの開発者が携わることから，適材適所の人材を探し出すこ

とは容易でない．このテーマでは，特定のソースコードや，バグ修正依頼などに対し，最も適切と思われる開発者を推薦する [30] [56].

[コミッター推薦] OSS プロジェクトでは，高い技術力や開発者を取りまとめる能力をもっていると認められた開発者のみ，プロダクトに加えた変更を構成管理システムにコミットするための権限 (コミット権限) が与えられる．このような開発者はコミッターと呼ばれ，プロジェクトに参加する一般開発者の中から適宜採用することが求められる．しかし，大規模プロジェクトでは一般開発者は 1 万人を超えることも珍しくないことから，コミッター候補を探し出すことは容易でない．本テーマでは，過去の機能拡張や不具合修正に関する活動記録を分析し，コミッター候補者を推薦することを目的としている [22].

[コミュニケーション分析] 従来，主に社会学の分野において，人と人との関係をネットワーク構造として捉えて分析するソーシャルネットワーク分析 (Social Network Analysis; SNA) が行われてきたが，本テーマでは，SNA を OSS コミュニティのコミュニケーションの分析に用いる．分析例としては，メーリングリストや掲示板におけるコミュニケーションを分析することで，開発者，ユーザ，バグ報告者の 3 者間を橋渡しする人物 (コーディネータ) がコミュニケーションを活発化させていることが明らかにされている [23]．また，近年，ソフトウェア開発 (の一部) を海外に発注する，いわゆるオフショア開発が増加し，コミュニケーションの問題が重要となっていることから，発注者と開発者間のコミュニケーションの分析も行われている [42].

[Openness (開放性) の分析] OSS 開発は，開発に大きな貢献を果たすいわゆるコアメンバーが中心となって進められるが，一般開発者であるノンコアメンバーからの patch 投稿を積極的に受け入れることも重要である．ただし，ノンコアメンバーに対してどれだけ開かれているか (opennesss) は，プロジェクトによって大きくことなり，閉鎖的なプロジェクトも存在する．本テーマでは，patch の受理状況を分析することで，プロジェクトの openness を評価している [54] [68].

3.4 OSS を利用する開発者の支援

近年，ソフトウェア開発の一部に OSS を利用するケースが増えており，OSS 利用者を支援することも重要なテーマとなっている．

[ライブラリの使用状況の分析] OSS を利用する企業では，どのバージョンのプロダクトを利用すべきかを決めることが重要である．古いバージョンは安定しているかもしれないが，機能は劣っている可能性がある．一方，新しいバージョンを採用するにも不安がある．本テーマでは，様々な OSS について，どのバージョンがもっとも使われているかを分析している [44]．ユーザ数の多いバージョンほど，機能と信頼性のバランスが取れていることになる．

[ソフトウェア部品検索] OSS は莫大な数が存在することから，利用者の必要とするソフトウェア部品 (ソースコード) を探し出すことは容易でない．そのために，ソフトウェア部品検索エンジンが多数開発されている．キーワードベースのサーチエンジンとしては，SPARS [59]，Koders [33]，Jarhoo [27] などがある．ソフトウェア部品の入出力をベースにした検索エンジンとしては，Prospector [38] がある．コードクローンに基づく検索ツールも提案されている [66]．検索結果のランキングの決定方法も重要な研究テーマとなっている [21].

[ソフトウェアの分類] 莫大な数の OSS を自動分類することで，類似ソフトウェアの発見に役立てようとする研究である．Kawaguchi ら [31] は，各 OSS プロダクトのソースコードから特徴語を抽出し，LSA(Latent Semantic Analysis; 潜在的意味解析手法) を用いて OSS を分類する方法を提案している．OSS 中で使用されているクラスファイルの違いに基づく分類も提案されている [62].

[ライセンスの分析] 本テーマは，テキストマイニング技術を用いて，OSS のライセンスを自動的に同定することを目的としている [37]．企業が OSS を利用してソフトウェアやサービスを開発・提供する場合，OSS の著作権者が定めたライセンスに従う必要がある．例えば，GPL(GNU General Public License)，BSD License，Apache License などである．ただし，OSS のライセンスは莫大な種類が存在し，1 つの OSS

プロダクトの中にそれぞれ異なるライセンスを持つ多数のサブプロダクトが含まれることも少なくないことから，OSS のライセンスを同定する技術が求められている．

[コーディングパターン検出] このテーマでは，ソースコードもしくはその履歴から，関数や API の使い方のパターンを抽出する [64] [67]．例えば，関数 A の後には関数 B が必ず呼ばれる，というようなパターンである．得られたパターンは，将来の開発や保守に役立てられる．近年では，シーケンシャルパターンマイニングを用いた手法が提案されている [25]．

3.5 MSR 研究者の支援

リポジトリマイニングを行う研究者を支援することも重要なテーマとなっている．Mockus は，マイニング対象のデータの取得と整形 (クリーニング) を行うのに全労力の 95%を要し，分析はわずか 1〜5%であると述べている [46]．このことから，マイニングを容易にするための技術が求められている．

[リポジトリ間のリンク付け] 本テーマは，異なるリポジトリに含まれる関連情報をリンク付けすることを目的としており，特に，バグ管理システム上のバグ修正情報と構成管理システム上のソースコード (の変更) とをリンク付けすることが盛んに研究されている．例えば，3.1 節で述べたバグモジュール予測の研究を行う場合，それぞれのバグがいつどのソースコードに混入し，いつ除去されたのかのデータが必要となる．リンク付けの方法として，最も基本的な方法が Gyimothy ら [13] によって提案され，その後，SZZ アルゴリズム [58]，および，その改良方法 [32] が提案されてきた．現在も研究は盛んにおこなわれており，Relink と呼ばれる方法が提案されている [65]．

[名寄せ] OSS 開発者は，リポジトリや作業内容に応じて，異なるアカウント名や email アドレスを使うことが少なくない．そのため，開発者に関する分析を行う前に，同一人物を特定してマージする「名寄せ」を行うことが必要である [4]．具体的な方法を 4.3 節で紹介する．

[MSR のサービス化] 誰でも簡単にリポジトリマイニングを行えるようにするために，MSR を Web サービス化することが提案されている [40]．本提案では，リポジトリとサービスがクラウド化されており，メトリクス計測サービス API，及び，統計分析サービス API を通してマイニングを行うことを目指している．

4 マイニングの具体例

3 章のいずれの研究においても，リポジトリからデータを取得し，必要な情報を抽出して整形する作業が必要不可欠である．ただし，従来の多くの論文では，紙面数の制約により，これら作業の詳細が記載されないことも多い．そこで，本章では，データ取得の例と整形作業の例 (“名寄せ”アルゴリズム) を紹介する．

4.1 データ取得方法

MSR 分野では，マイニング目的を決定後，どのリポジトリからどのようなデータを取得するかを検討する．例えば，最近エンピリカルソフトウェア工学分野のトップカンファレンスで注目されている研究の 1 つである bug localization(プロジェクトに報告されたバグ票のテキスト情報に基づき，欠陥が混入していると思われるソースコードの特定) に関する研究 [70] では，欠陥を含むソースコードを探すための手がかりを得るために，バグ管理システムからバグを再現する情報等を取得する．また，バグが報告された時点のソースコードを，ソフトウェア構成管理システムからチェックアウトする必要がある．一方，開発者の協調作業，共同作業の関係を明らかにするための研究では，1 つのアプローチとして SNA が挙げられ，メールアーカイブをダウンロードし，メールのヘッダ情報や内容を抽出する [41]．本節では，ソフトウェア構成管理システム，バグ管理システム，メーリングリストの各リポジトリからのデータ取得の一部を紹介する．

4.1.1 ソフトウェア構成管理システム

多くの研究では，プロジェクトが管理するソフトウェア構成管理システムに保存されている開発記録を，Web を介して取得している．

コミットログには，ソースコードを変更した記録が残されており，変更バージョン (リビジョン) 名，変

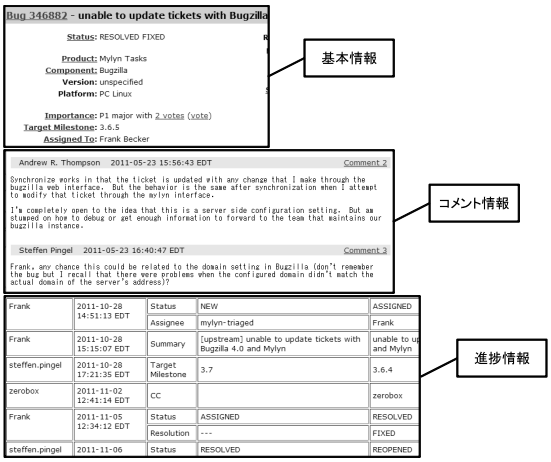

図 2 バグ管理システム Bugzilla のバグ票

更日時，変更者名，変更行数 (追加行数，削除行数)，変更理由が記録されている．コミットログから正規表現などを用いて，目的に応じて必要な情報を抽出する．

多くのオープンソースソフトウェア開発で使用されている Subversion は，中央集中型ソフトウェア構成管理システムである．これらのリポジトリでは，1 つのリポジトリを多人数で共有するため，誰もが最新バージョンを取得できる．しかしながら，ある開発者がソースコードを編集している間は，その他の開発者は当該ソースコードを編集することはできない．この問題を解決する新たなリポジトリとして分散型ソフトウェア構成管理システム Git が開発された．Git は開発者が各自のリポジトリにコミットし，一区切りついた時点で他人との差分を取得し，自身が変更した内容とマージされる．新たなソフトウェア構成管理システムとして，現在 MSR 分野で注目されている [5].

4.1.2 バグ管理システム

バグ管理システムは，開発者あるいはユーザから報告されたバグ情報をバグ票に記録し，開発者がバグ修正の進捗を記録しながら，修正の様子を共有するシステムである．図 2 は，多くのオープンソースプロジェクトで利用されているバグ管理システム Bugzilla に記録されている各情報である．図に示されるように，バグ票には 3 種類の記録が残される．1 つはバグ報告時に記録される基本情報 (バグが発見された機能名，報告者名，優先度，再現方法等)，開発者が当該バグの修正に向けて議論するコメント情報，最後に，バグ修正中に記録されるバグの進捗情報である．これらの情報は，Web インタフェースを介して誰でも閲覧することができる．

バグ管理システムに登録されたバグ票の情報を取得するために，まずバグ票の HTML を取得する．多くのプログラミング言語が HTML を取得するためのライブラリをサポートしている．取得した HTML ファイルから，正規表現などを用いてタグを削除し，目的に応じて必要な情報 (例えば，バグ報告日時，バグ混入モジュール名，修正担当者など) を抽出する．

4.2 メーリングリストアーカイブ

大規模ソフトウェア開発では，複数の開発者が情報を共有するために，メーリングリストが使用される．特にオープンソース開発プロジェクトでは，プロジェクトの課題，開発計画等をプロジェクト全体に認知させることを目的として，メーリングリストが使用されている．メールは基本的にヘッダとボディ (メール本文) に分かれている．ヘッダには，メール ID，送信者，送信日時，受信者，返信元メール ID，題名などが記録されている．

メーリングリストを用いたその他のマイニング対象として添付ファイルが挙げられる．プロジェクトによって方針が異なるが，patch をメールに添付して，プロジェクトで確認される場合があるため，添付ファイルの有無などを分析している研究もある．

4.3 データ整形方法

各ソフトウェアリポジトリからデータをダウンロードした後に，分析を容易にするフォーマットに書き換える必要がある．開発者はリポジトリ，開発環境，実施する作業の役割に応じてアカウント名や email アドレスを変えることがある．例えば，版管理システムが提示する開発者名はアカウント名であり，メーリングリストで提示されるのは email アドレスである．このような場合に，Christian Bird が提案した“名寄せ”アルゴリズムがある [4]．名寄せアルゴリズムは，類似するアカウント，email アドレスが同一人物

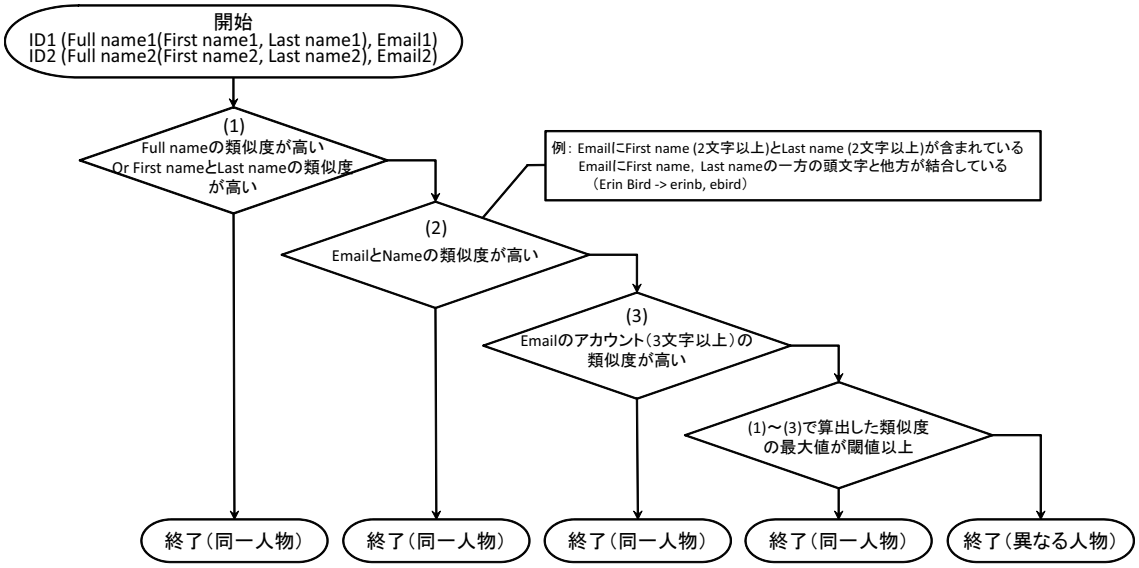

図 3 “名寄せ” アルゴリズム

のものか否かを判断するためのアルゴリズムである．具体的なアルゴリズムは図 3 の通りであり，基本的な内容としては，fullname，first name，last name，email アドレスのアカウント名の類似度合いをレーベンシュタイン距離を用いて算出し，閾値以上であれば同一人物と判断する．論文では明確な閾値は提示されておらず，完全に一致させることは難しいが，現在提案されている方法の中で最も信頼性の高いアルゴリズムであるため数多くの研究で使用されている．

これ以外の整形技術として，バグモジュールを予測するために高品質なモデル構築に向けたデータセットの作成技術 (欠損値補完技術，名義尺度のダミー変数化)，テキストデータを用いた分析を行うための形態素解析技術などがある．リポジトリから抽出したデータをそのまま使用できることは少なく，多くの場合，データ整形が行われる．データ整形は，地味な作業かもしれないが，データ整形を行うことでより正しい結果，より良い結果を得ることができる．

4.4 データ取得後のプロセス

データ取得・整形後は，データ処理プロセス (Processing) と分析プロセス (A nalysis) が続く [28]．データ処理プロセスはデータの種類ごとに異なり，例えば，ソースコードについては抽象構文木への変換，トークンの置き換え，バージョン間の差分の抽出，メトリクス計測などが行われる．テキストデータについては，トークン化，ストップワード除去，語幹処理 (stemming)，グラフ表現やベクトル表現への変換などが行われる．また，分析プロセスでは，分類，回帰，ルール抽出，クラスタリングのための様々なデータマイニング技術が用いられる．詳しくは，Jung らのサーベイ論文 [28] を参照されたい．

5 MSR の発展

現在，100 人以上の参加者が集まる MSR Conference は，1 年に 1 度の国際会議にとどまらず，MSR 研究者の増加，そして，MSR 分野の発展に向けて，MSR committee らが精力的に活動している．2010 年には，世界中の学生へ MSR の最新動向，及び，マイニング技術を合宿形式で解説する Mining Summer School 2010 がカナダで開催された．その内容は，頻繁に引用される論文の紹介，リポジトリマイニングを対象とした統計解析ツール R やデータマイニングツール Weka の使い方，論文の書き方等が解説され，MSR 分野における効率的な実験，および，質の高い論文の執筆を支援することを目的としている．

2012 年 8 月には，100 人を超える世界中の MSR 研究者が集い，2020 年に向けて何にチャレンジしていくべきかを議論する MSR vision 2020 がカナダで開催された [45]．参加者が 15 人程度の小グループに分かれて，現在の MSR 分野が成功した要因，失敗していること，2020 に向けて注目される研究トピックについてオープンディスカッションを行った．以降，MSR vision で議題となった MSR 研究トピックの一部 (e.g. Large/Small-Scale mining, IR, Github,Visualization) を紹介する．

- Large/Small-Scale mining: ICSE や MSR に投稿される論文において実験対象となっているのは大規模プロジェクトである．大規模プロジェクトには解決すべき多くの課題を有しているが，大規模プロジェクトには多くの小規模プロジェクトが内在しており，大規模プロジェクトだけでなく小規模プロジェクトを対象とした研究も重要ではないかという議論が行われた．小規模プロジェクトを対象とした場合に課題になってくることの 1 つとして，リポジトリをマイニングするに足る十分な情報が記録されているか否かが 1 つの課題になる．
- IR (Information retrieval): 近年，ビッグデータの解析に関する研究が注目されており，IR はソフトウェア工学に限らず，ビッグデータの中から目的に合致したものを取り出すことである．これまでもバグ票，コミットログ，メーリングリストを対象とした IR 技術の研究が提案されているが，IR 技術の役に立つ情報，役に立たない情報があり，MSR vision では，日々ソフトウェア開発データを解析している研究者で情報共有を行った．例えば，IR 技術ではバグ票のタイトルは役に立つが，その他のテキストはノイズが多く，検索・予測の精度をむしろ低下させる場合がある，といった意見があった．
- Github: Github はソフトウェア構成管理システムの一種であり， Git のリポジトリをホスティングするサービスである．Github はソフトウェア構成管理システムだけでなく，wiki やタスク管理ツールなど，開発者間のコラボレーションをサポートする機能を提供しているため，Github を使用するプロジェクトが増加している．Git, Github は個人のリポジトリを保有することができるため，従来までの CVS や Subversion のように全てのリポジトリを開発者全員で共有する場合とは異なる．リポジトリ方式が変われば，それに従ってマイニング技術も異なってくる．近年 Dr. Daniel German 等，数多くの研究者が注目している．

このように，将来の MSR 研究者の育成，及び，MSR 分野の発展に向けた活動が積極的に行われている．

日本でも，若手研究者が中心となって，MSR の Steering Committee や MSR 分野で活躍する研究者を招待して講義する MSR School in Asia を 2010 年より毎年開催している．これまでに招待された研究者は錚々たるメンバーである．

2010 年: Dr. Ahmed E. Hassan,
Dr. Thomas Zimmermann,
Dr. Sunghun Kim
2011 年: Dr. Tao Xie，Dr. Tien N. Nguyen
2012 年: Dr. Daniel German

MSR School in Asia に興味を持った方はぜひ参加を検討いただけたらと思う．

6 まとめ

本稿では，MSR 研究の魅力に焦点を当て，主な研究テーマを紹介するとともに，マイニングの具体例と MSR 分野の今後の発展について述べた．

近年，Git や Trac などの新たなリポジトリが登場し，新たな OSS も日々登場していることから，MSR 研究はますます盛んになっていくと期待される．研究テーマを広げていく 1 つの方向性は，マイニングの対象を単一バージョンのプロジェクトから，複数バージョンやプロダクトファミリー [11] へと広げることである．また，異なるプロダクト間の比較を行い，マイニングにより得られた知見の一般性を評価することも，今後重要となる．

参考文献

[1] Anvik, J., Hiew, L. and Murphy, G.C.: Who should fix this bug?, in *Proc. 28th International Conference on Software Engineering* (*ICSE'06*), pp. 361–370.

[2] Aman, H., Mochiduki, N. and Yamada, H.: A model for detecting cost-prone classes based on Mahalanobis-Taguchi method, *IEICE Trans. on Inf. & Syst.*, Vol. E89-D, No. 4 (2006), pp. 1347–1358.

[3] Askari, M. and Holt, R.: Information theoretic evaluation of change prediction models for large-scale software, in *Proc. 2006 International Workshop on Mining Software Repositories* (*MSR'06*), pp. 126–132.

[4] Bird, C., Gourley, A., Devanbu, P., Gertz, M. and Swaminathan, A.: Mining Email Social Networks, in *Proc. 3rd International Workshop on Mining Software Repositories* (*MSR'06*), 2006, pp. 137–143.

[5] Bird, C., Rigby, C. P., Barr, T. E., Hamilton, J. D., German, M. D. and Devanbu, P.: The promises and perils of mining git, in *Proc. 6th International Workshop on Mining Software Repositories* (*MSR'09*), 2009, pp. 1–10.

[6] Caglayan, B., Bener, A. and Koch, S.: Merits of using repository metrics in defect prediction for open source projects, in *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development* (*FLOSS 2009*), 2009, pp. 31–36.

[7] Canfora, G. and Cerulo, L.: Fine grained indexing of software repositories to support impact analysis, in *Proc. 3rd International Workshop on Mining Software Repositories* (*MSR'06*), 2006, pp. 105–111.

[8] D'Ambros, M., Lanza, M. and Robbes, R.: Evaluating defect prediction approaches: a benchmark and an extensive comparison, *Empirical Software Engineering*, Vol. 17, Issue 4–5 (2012), pp. 531–577.

[9] 独立行政法人情報処理推進機構: 第 3 回オープンソースソフトウェア活用ビジネス実態調査, 2010.

[10] Fischer, M., Pinzger, M. and Gall, H.: Populating a release history database from version control and bug tracking systems, in *Proc. 19th IEEE International Conference on Software Maintenance* (*ICSM'03*), 2003, pp. 23–32.

[11] Fischer, M., Oberleitner, J., Ratzinger, J. and Gall, H.: Mining evolution data of a product family, in *Proc. 2nd International Workshop on Mining Software Repositories* (*MSR'05*), 2005, pp. 12–16.

[12] Fluri, B., Wursch, M., Pinzger, M. and Gall, H. C.: Change Distilling:Tree Differencing for Fine-Grained Source Code Change Extraction, *IEEE Transactions on Software Engineering*, Vol. 33, Issue 11 (2007), pp. 725–743.

[13] Gyimothy, T., Feremc, R. and Siket, I.: Empirical validation of object-oriented metrics on open source software for fault prediction, *IEEE Trans. Soft. Eng.*, Vol. 31, No. 10 (2005), pp. 897–910.

[14] 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 (2000), pp. 653–661.

[15] Hassan, A. E.: The road ahead for Mining Software Repositories, in *Frontiers of Software Maintenance* (*FoSM*) *at the 24th IEEE International Conference on Software Maintenance* (*ICSM2008*), 2008, pp. 48–57.

[16] Hassan, A. E. and Holt, R. C.: The top ten list: Dynamic fault prediction, in *Proc. ICSM 2005*, 2005, pp. 263–272.

[17] Hassan, A. E.: Predicting faults using the complexity of code changes, in *Proc. ICSE 2009*, 2009, pp. 78–88.

[18] 畑秀明, 水野修, 菊野亨: 不具合予測に関するメトリクスについての研究論文の系統的レビュー, コンピュータソフトウェア, Vol. 29, No. 1 (2012), pp. 106–117.

[19] 林卓磨，信田祥司，白井久美子，吉田順一，門田暁人，松本健一: Fault-prone モジュール判別モデルを用いた検証項目数削減に関する一検討，2011 年電子情報通信学会ソサイエティ大会通信講演論文集 2，B-18-2, 2011, p. 406.

[20] Hindle, A., German, D. M. and Holt, R.: What do large commits tell us?: a taxonomical study of large commits, in *Proc. 2008 International Working Conference on Mining Software Repositories* (*MSR'08*), 2008, pp. 99–108.

[21] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations, *IEEE Transactions on Software Engineering*, Vol. 31, No. 3 (2005), pp. 213–225.

[22] 伊原彰紀, 亀井靖高, 大平雅雄, 松本健一, 鵜林尚靖: OSS プロジェクトにおける開発者の活動量を用いたコミッター候補者予測, 電子情報通信学会論文誌, Vol. J95-D, No. 2 (2012), pp. 237–249.

[23] 伊原彰紀, 大平雅雄, まつ本真佑, 亀井靖高, 松本健一：複数のサブコミュニティを有する OSS コミュニティを対象としたネットワーク分析, 情報処理学会 マルチメディア，分散，協調とモバイル (DICOMO2008) シンポジウム, 2008, pp. 297–303.

[24] Ihara, A., Ohira, M. and Matsumoto, K.: An Analysis Method for Improving A Bug Modification Process In Open Source Development, in *Proc. 10th international workshop on principles of software evolution* (*IWPSE 2009*), 2009, pp. 135–143.

[25] 石尾隆, 伊達浩典, 三宅達也, 井上克郎: シーケンシャルパターンマイニングを用いたコーディングパターン抽出, 情報処理学会論文誌, Vol. 50, No. 2 (2009), pp. 860–871.

[26] Jang, J., Agrawal, A. and Brumley, D.: ReDeBug: Finding Unpatched Code Clones in Entire OS Distributions, in *Proc. 33rd IEEE Symposium on Security and Privacy*, 2012, pp. 48–62.

[27] Jarhoo, http://www.jarhoo.com/

[28] Jung, W., Lee, E. and Wu, C.: A Survey on

Mining Software Repositories, *IEICE Trans. on Information and Systems*, Vol. E95-D, No. 5 (2012), pp. 1384–1406.

[29] Kagdi, H., Collard, M. L. and Maletic, J. I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution, *J. Software Maintenance and Evolution: Research and Practice*, Vol. 19, No. 2 (2007), pp. 77–131.

[30] Kagdi, H. and Poshyvanyk, D.: Who can help me with this change request?, in *Proc. 17th International Conference on Program Comprehension* (*ICPC 2009*), 2009, pp. 273–277.

[31] Kawaguchi, S., Garg, P. K., Matsushita, M. and Inoue, K.: MUDABlue: An Automatic Categorization System for Open Source Repositories, *Journal of Systems and Software*, Vol. 79, No. 7 (2006), pp. 939–953.

[32] Kim, S., Zimmermann, T., Pan, K. and Whitehead, E. J. Jr.: Automatic identification of bug-introducing changes, in *Proc. 21st International Conference on Automated Software Engineering* (*ASE2006*), 2006, pp. 81–90.

[33] Koders.com, http://www.koders.com/

[34] 纐纈伸子, 川村真弥, 野村准一, 野中誠: プロセスおよびプロダクトメトリクスを用いた fault-prone クラス予測の適用事例, 情報処理学会研究報告, ソフトウェア工学研究会, Vol. 2010-SE-168, No. 6 (2010).

[35] Livieri, S., Higo, Y., Matsushita, M. and Inoue, K.: Analysis of the Linux Kernel Evolution Using Code Clone Coverage, in *Proc. 4th International Workshop on Mining Software Repositories* (*MSR2007*), 2007, pp. 22-1–4

[36] Livshits, B. and Zimmermann, T.: DynaMine: Finding common error patterns by mining software revision histories, in *Proc. 13th International Symposium on Foundations of Software Engineering* (*ESEC/FSE'05*), 2005, pp. 296–305.

[37] 真鍋雄貴, ダニエル モラレスゲルマン, 井上克郎: 階層的ライセンス知識を用いたライセンス特定ツールの開発, 情報処理学会論文誌, Vol. 52, No. 8 (2011), pp. 2402–2411.

[38] Mandelin, D., Xu, L., Bodik, R. and Kimelman, D.: Jungloid mining: helping to navigate the API jungle, in *Proc. Programming Language Design and Implementation* (*PLDI 2005*), 2005, pp. 48–61.

[39] Matsumoto, S., Kamei, Y., Monden, A., Matsumoto, K. and Nakamura, M.: An Analysis of Developer Metrics for Fault Prediction, in *Proc. PROMISE 2010*, 2010.

[40] Matsumoto, S. and Nakamura, M.: Service Oriented Framework for Mining Software Repository, in *Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement* (*IWSM/MENSURA 2011*), 2011, pp. 13–19.

[41] 柗本真佑, 亀井靖高, 大平雅雄, 松本健一: OSS コミュニティにおけるオープンコラボレーションの理解, 情報社会学会誌, Vol. 3, No. 2 (2009), pp. 29–42.

[42] 松村知子, 吉田誠, 井手直子, 森崎修司, 戸田航史, 松本健一: ソフトウェア開発の要件定義工程におけるユーザ・ベンダ間のコミュニケーション分析と活用方法, プロジェクトマネジメント学会 2011 年度 春季研究発表大会予稿集, 2011, pp. 427–432.

[43] 松下誠: ソフトウェア工学の新潮流 (1) リポジトリマイニング, ソフトウェアエンジアリング最前線 2009, pp. 21–24.

[44] Mileva, Y. M., Dallmeier, V., Burger, M. and Zeller, A.: Mining Trends of Library Usage, in *Proc. Joint International and Annual ERCIM Workshops on Principles of Software Evolution* (*IWPSE*) *and Software Evolution* (*Evol*) *Workshops*, 2009, pp. 57–62.

[45] MITACS MSR vision 2020: http://msrcanada.org/msrvision2020/, 2012.

[46] Mockus, A.: How to run empirical studies using project repositories, in *4th International Advanced School of Empirical Software Engineering* (*IASESE 2006*), 2006.

[47] Monden, A., Okahara, S., Manabe, Y. and Matsumoto, K.: Guilty or Not Guilty: Using Clone Metrics to Determine Open Source Licensing Violations, *IEEE Software*, Vol. 28, No. 2 (2011), pp. 42–47.

[48] 門田暁人, 伊原彰紀: リポジトリマイニングの研究成果の産業化への応用, ウィンターワークショップ 2013・イン・那須, 2013.

[49] Morisaki, S., Matsumura, T., Ohkura, K., Fushida, K., Kawaguchi, S. and Iida, H.: Fine-Grained Software Process Analysis to Ongoing Distributed Software Development, in *Proc. 1st Workshop on Measurement-based Cockpits for Distributed Software and Systems Engineering Projects* (*SOFTPIT 2007*), 2007, pp. 26–30.

[50] 大平 雅雄: ソフトウェア開発現場での活用へ向けたソフトウェアリポジトリマイニング手法の体系化, 情報処理学会シンポジウムシリーズ, ウインターワークショップ 2012・イン・琵琶湖, Vol. 2012, No. 1, pp. 49–50.

[51] 大平雅雄, イェ ユンウェン, 中小路久美代, 山本恭裕: ソフトウェア開発における知識コラボレーション, 人工知能学会誌, Vol. 26, No. 1 (2011), pp. 66–78.

[52] Olatunji, S. O., Idrees, S. U., Al-Ghamdi, Y. S. and Al-Ghamdi, J. S. A.: Mining Software Repositories – A Comparative Analysis, *Int'l J. Computer Science and Network Security*, Vol. 10, No. 8 (2010), pp. 161–174.

[53] Phannachitta, P., Ihara, A., Jirapiwong, P., Ohira, M. and Matsumoto, K.: An Algorithm for Gradual Patch Acceptance Detection in Open Source Software Repository Mining, *IEICE Transactions on Information and Systems*, Vol. E95-A, No. 9 (2012), pp. 1478–1489.

[54] Phannachitta, P., Jirapiwong, P., Ihara, A., Ohira, M. and Matsumoto, K.: Understanding Oss Openness Through Relationship between Patch Acceptance and Evolution Pattern, in *Proc. International Workshop on Empirical Software Engineer-*

ing in Practice (*IWESEP2011*), 2011, pp. 37–42.

[55] Purushothaman, R. and Perry, D. E.: Toward understanding the rhetoric of small source code changes, *IEEE Transactions on Software Engineering*, Vol. 31, No. 6 (2005), pp. 511–526.

[56] Servant, F. and Jones, J. A.: WhoseFault: Automatic Developer-to-Fault Assignment Through Fault Localization, in *Proc. 34th International Conference on Software Engineering* (*ICSE2012*), 2012, pp. 36–46.

[57] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E. and Matsumoto, K.: Studying Re-Opened Bugs in Open Source Software, Empirical Software Engineering, (available online Sep. 2012), http://link.springer.com/article/10.1007%2Fs10664-012-9228-6

[58] Sliwerski, J., Zimmermann, T. and Zeller, A.: When do changes induce fixes? in *Proc. MSR'05*, 2005, pp. 1–5.

[59] SPARS Project, http://sel.ist.osaka-u.ac.jp/SPARS/

[60] StagE プロジェクト::エンピリカルデータに基づくソフトウェアタグ技術の開発と普及, http://www.stage-project.jp/

[61] Tsunoda, M., Matsumura, T., Iida, H., Kubo, K., Kusumoto, S., Inoue, K. and Matsumoto, K.: Standardizing the Software Tag in Japan for Transparency of Development, in *PROFES 2010*, LNCS 6156, Springer, 2010, pp. 220–233.

[62] 牛窓朋義, 門田暁人, 玉田春昭, 松本健一: 使用クラスに基づくソフトウェアの機能面からの分類, 電子情報通信学会技術報告, ソフトウェアサイエンス研究会, Vol. 109, No. 170 (2009), pp. 31–36.

[63] Weiss, C., Premraj, R., Zimmermann, T. and Zeller, A.: How long will it take to fix this bug? in *Proc. 4th International Workshop on Mining Software Repositories* (*MSR'07*), 2007, pp. 1–8.

[64] Williams, C. C. and Hollingsworth, J. K.: Recovering system specific rules from software repositories, in *Proc. 2nd International Workshop on Mining Software Repositories* (*MSR'05*), 2005, pp. 7–11.

[65] Wu, R., Zhang, H., Kim, S. and Cheung, S.C.: Relink: recovering links between bugs and changes, in *Proc. 19th ACM SIGSOFT Symposium and 13th European Conference on Foundations of Software Engineering* (*ESEC/FSE 2011*), 2011, pp. 15–25.

[66] Xia, P., Manabe, Y., Yoshida, N. and Inoue, K.: Development of a code clone search tool for open source repositories, コンピュータソフトウェア, Vol. 29, No. 3 (2012), pp. 181–187.

[67] Xie, T. and Pei, J.: MAPO: Mining API usages from open source repositories, in *Proc. 3rd International Workshop on Mining Software Repositories* (*MSR'06*), 2006, pp. 54–57.

[68] Yamamoto, M., Ohira, M., Kamei, Y., Matsumoto, S. and Matsumoto, K.: Temporal Changes of the Openness of An Oss Community: A Case Study of the Apache Http Server Community, in *Proc. 5th International Conference on Collaboration Technologies 2009* (*CollabTech 2009*), 2009, pp. 64–65.

[69] Ying, A. T. T., Murphy, G. C., Ng, R. and Chu-Carroll, M. C.: Predicting source code changes by mining change history, *IEEE Transactions on Software Engineering*, Vol. 30, No. 9 (2004), pp. 574–586.

[70] Zhou, J., Hongyu Zhang, H. and Lo, D.: Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports, in *Proc. 34th International Conference on Software Engineering* (*ICSE2012*), 2012, pp. 14–24.

[71] Zimmermann, T., Weisgerber, P., Diehl, S. and Zeller, A.: Mining version histories to guide software changes, in *Proc. 26th International Conference on Software Engineering* (*ICSE'04*), 2004, pp. 563–572.

[72] Zimmermann, T., Nagappan, N., Gall, H., Giger, E. and Murphy, B.: Cross-project defect prediction—A Large Scale Experiment on Data vs. Domain vs. Process, in *Proc. 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* (*ESEC/FSE'09*), 2009, pp. 91–100.

門田暁人

1994 年名古屋大学工学部電気学科卒業．1998 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年同大学同研究科助手．2004 年同大学助教授．2007 年同大学准教授．2003〜2004 年 Auckland 大学客員研究員．博士 (工学)．ソフトウェアメトリクス，ソフトウェアプロテクション，ヒューマンファクタ等の研究に従事．日本ソフトウェア科学会，情報処理学会，電子情報通信学会，IEEE，ACM 各会員．

伊原彰紀

2007 年龍谷大学理工学部卒業．2009 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．2012 年同大学博士課程修了．同年同大学情報科学研究科・助教．博士（工学）．ソフトウェア工学，特にオープンソースソフトウェア開発・利用支

援の研究に従事．日本ソフトウェア科学会，情報処理学会，電子情報通信学会，IEEE 各会員．

松 本 健 一

1985 年大阪大学基礎工学部情報工学科卒業．1989 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．1993 年奈良先端科学技術大学院大学助教授．2001 年同大学教授．工学博士．エンピリカルソフトウェア工学，特に，プロジェクトデータ収集/利用支援の研究に従事．日本ソフトウェア科学会，情報処理学会，電子情報通信学会，ACM 各会員，IEEE Senior Member．