様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

令和　３ 年　 ６ 月 １８ 日現在

機関番号：１４７０１
研究種目：基盤研究(B)（特設分野研究）
研究期間：2018～2020
課題番号：１８ＫＴ００１３

研究課題名（和文）超長寿命ソフトウェアを実現する自律的な保守性強化技術の研究

研究課題名（英文）A study on autonomous maintainability technique toward ultra long-term software

研究代表者
伊原　彰紀（Ihara, Akinori）

和歌山大学・システム工学部・講師

研究者番号：４０６３８３９２

交付決定額（研究期間全体）：（直接経費）　　12,900,000円

研究成果の概要（和文）：本研究課題は，超長寿命ソフトウェアの実現に向けて，ソーシャルコーディングにおいてソフトウェア品質を強化する開発者らに共通する実装方法を明らかにし，それらをルールとして形式化する手法を開発した．さらに，組織の習慣，技術の動向に合わせて自律的にルールを更新する手法を開発し，継続的なソフトウェア保守技術を確立した．具体的には，ソフトウェア部品単位の理解容易性を高めるための局所的な保守性の強化技術，また，設計されたソフトウェアの全体構造を維持するための大域的な保守性の強化技術を提案した．

研究成果の学術的意義や社会的意義
本研究課題が対象とするソフトウェアの改善提案は，検証前のソースコード，言い換えると，下書き段階のソースコードであり，ソフトウェアに採用されなかったソースコードも含む．ソフトウェア工学分野における多くの研究は，ソフトウェアのリリース後のソフトウェア部品の修正を対象としているため，本研究課題の学術的意義は高い．また，本研究課題は，Linuxのように人間の寿命を超えて開発が継続する超長寿ソフトウェアの実現に向けて，開発者が入れ替わり，新たな機能，技術が導入される中でもソフトウェア製品の保守作業の指針を柔軟に改定することで高い品質を維持する技術であり，社会的意義が高い．

研究成果の概要（英文）：This research project clarified a common implementation technique among developers toward maintaining software project for ultra long-term, and developed an approach to formalize the implementation rules. Furthermore, this project developed an approach to autonomously updating the rules as a continuous maintenance technology. Finally, this project released an automatic code review system "DevReplay" containing a local maintenance technology and a global maintenance technology.

研究分野：ソフトウェア工学

キーワード：コードレビュー　ソフトウェア保守　ソーシャルコーディング　プログラム解析　コーディング規約

様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９（共通）

１．研究開始当初の背景

金融，流通などのミッションクリティカルなシステムに使用される Linux は，1973 年に C 言語で実装されてから 48 年余りが経過した．Linux の開発が現在も進められる背景には，ソースコードがオープンソースソフトウェア（OSS）として広く公開されることで，不特定多数の開発者が高品質なソフトウェアを実現するためにソフトウェア部品の改善を提案することが可能となり，開発者が入れ替わりながらも停止することなく保守され続けているという状況がある．Linux は，今日も商用，非商用に関わらず多くのシステムを支える長寿命ソフトウェアとして成長を続けている．このような長寿命ソフトウェアを生み出すために，ソフトウェア開発企業の Microsoft，Facebook 等の大企業でさえも，ソフトウェア部品を広く公開し，クラウド上で他者と共同でソフトウェアの新機能を実装，提案する開発形態（ソーシャルコーディング）を導入している．ソーシャルコーディングは，ソフトウェアの開発データを管理する共有 Web サービス GitHub が想定する開発形態であり，ソフトウェア開発のデファクトスタンダードとして確立されつつある．GitHub は，SNS 機能によってソフトウェア部品を構成管理するデータサーバの共有，伝搬を容易にしている．構成管理のためのサーバを誰もが保有することができるようになったことで，ソフトウェア開発への参加間口が拡大し，他の開発者が実装した機能を受け取ることができるようになった．

今日のソーシャルコーディングは，多くの開発リソースの確保を可能にする一方で，実装スタイルの異なる個々の提案を保守するためのコストが増大するジレンマを引き起こしている [1]．ソーシャルコーディングにおけるジレンマを解決するためには，ソフトウェアの実装方法の共通化，すなわち，共同開発する開発者が相互に期待する保守性の高いソフトウェアの実装方法を共有することが必要である．

２．研究の目的

本研究課題は，超長寿命ソフトウェアの実現に向けて，ソーシャルコーディングにおいてソフトウェア品質を強化する開発者らに共通する実装方法を明らかにし，それらをルールとして形式化する手法を開発した．さらに，組織の習慣，技術の動向に合わせて自律的にルールを更新する手法を開発し，継続的なソフトウェア保守技術を確立した．具体的には，ソフトウェア部品単位の理解容易性を高めるための局所的な保守性の強化技術，また，設計されたソフトウェアの全体構造を維持するための大域的な保守性の強化技術を提案した．

３．研究の方法

ソフトウェア中の欠陥の発見，ソースコードの可読性を改善するために，開発者は自身の勘や経験によってソースコードの局所的な改善，大域的な改善を行なっている．

- 局所的な改善：ソースコードの可読性や保守性の強化
- 大域的な改善：設計されたモジュール構造の改善，モジュール単位での理解容易性の強化

これらの改善は，ソースコードの読みやすさ，機能追加の容易性，ソースコード変更の局所化，欠陥混入の防止に貢献する．本研究課題では，これらの改善方法をソフトウェアの保守性を維持するための規約として形式化することを実現した．本研究課題において，それぞれの保守性強化技術を述べる．

（１）　局所的な改善のための保守性強化技術

本研究課題では，局所的な改善のため保守性強化技術として，コードレビュー管理システムおよび GitHub において変更提案されたソースコード（パッチ）に対して，コードレビューによって行われた変更を分析した．パッチ開発者が作成し，プロジェクトに投稿したパッチを初版 Patch_1 とし，検証者による指摘とパッチ開発者による修正を繰り返してプロジェクトに採用されたパッチを最終版 Patch_n とする．本研究課題では，まずコードレビューによる変更内容は多岐にわたるため，パッチが対象プロジェクトに追加するソースコードのチャンク（断片）のうち，Patch_1 と Patch_n の間で変更されたもの（以降，変更チャンク）の内容を目視で分析した．変更チャンクは diff コマンドで出力したソースコードの差分から得られる追加行と削除行によって構成されている．このとき，文字の置き換えなどの変更は追加と削除の組み合わせによって表現した．次に，Patch_1 を適用した状態のソースコードと Patch_n を適用した状態のソースコードに対して静的解析ツールを実行し，コーディング規約違反に関する分析を行なった．また，本分析に基づき，開発者らに共通するソースコードを局所的に改善する実装方法を明らかにし，それらを正規表現によって形式化する手法を開発した．

（２）　大域的な改善のための保守性強化技術

本研究課題では，大域的な改善のため保守性強化技術として，コードレビューにおいて採択/改善要求/不採択されるパッチの変更内容の特徴を抽出する方法を提案した．特に変更内容の特徴として，パッチにおける振る舞い変化（メソッドレベルの変更内容）の有無を計測する．

また，本手法を用いて，コードレビュー対象のソースコードが採択/改善要求/不採択に影響するか否かを調査した．具体的には，変更前のソースコード Patch_0 と変更後のソースコード Patch_1 を対象に，いずれか一方でテストコードを生成し，他方に実行した結果の出力値を確認する．テストコードの実行結果が失敗（Failure）の場合，ソースコードの振る舞いが変わる変更が加えられたと捉える．

変更内容は，実装者が変更する前のソースコードと，変更した後のソースコードの振る舞いの違いとする．検証者がコードレビューを行う主な動機は，欠陥有無の確認である [2]．スペルやインデントの修正などの出力に影響しない変更に比べ，機能追加のような出力に影響する変更は，欠陥を混入しやすい．そのため，変更内容は検証者によるプルリクエストの受け入れに影響すると考えられる．本研究課題ではソースコードの振る舞いの異なる 4 種類の変更を計測する．

- Pattern1 [SS：Success-Success] － 2 種類のテスト結果がともに Success である．変更前と変更後のプログラムがともに同様の出力を行っているため，リファクタリングや実装方法の変更などの出力に影響を与えない変更が行われたと捉える．
- Pattern2 [SF：Success-Failure] － Patch_0 のテストケースを Patch_1 に実行した場合は Success，Patch_1 のテストケースを Patch_0 に実行した場合は Failure である．変更前の仕様を変更後でも満たすが，変更後の仕様は変更前では満たさないため，前方互換性のない変更であると捉える．
- Pattern3 [FS：Failure-Success] － Patch_0 のテストケースを Patch_1 に実行した場合は Failure，Patch_1 のテストケースを Patch_0 に実行した場合は Success である．変更後の仕様を変更前でも満たすが，変更前の仕様は変更後では満たさないため，後方互換性のない変更であると捉える．
- Pattern4 [FF：Failure-Failure] － 2 種類のテスト結果がともに Failure である．変更前の仕様を変更後で満たさず，変更後の仕様を変更前で満たさないため，互換性のない変更であると捉える．

４．研究成果

（1） 局所的な改善のための保守性強化技術

① ソースコード改善内容の調査

GitHub を用いて管理される大規模ソフトウェア OpenStack，Google，Microsoft，Facebook を対象に，コードレビューを通して改善されるソースコードの変更内容を調査した．Patch1 と Patch_n の変更対 384 件（許容誤差は 5%，信頼度は 95%）を対象に，既存研究 [3] が定義した不具合修正パターンに基づき変更内容がコード改善であるか否かを目視で検出した．本研究が分析対象としたソフトウェアは，Python 言語を主に使用しており，開発者が統一的な実装方法を実現するために，ソフトウェア開発組織でコーディング規約として，Python 言語標準のコーディング規約 PEP8 に基づいた静的解析ツール Pylint や Flake8 を使用している．プロジェクト内のソースコード記述方式に一貫性をもたせることを目的に静的解析ツールの使用を推奨しているが，コードレビューにおいて検証者は，コーディング規約以外についても確認している．

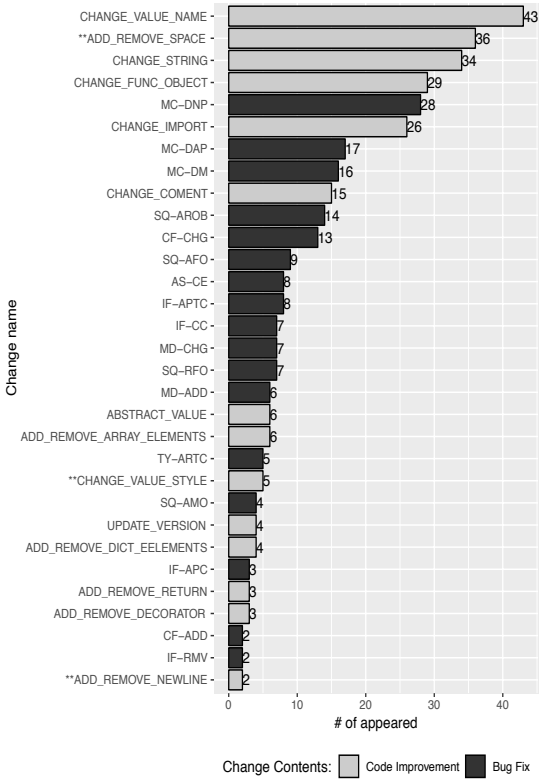

図 1 コードレビューを通して実施されたソースコード改善と不具合修正の出現回数（総変更数: 384 件中ソースコード改善は 211 件）

図 1 は，本研究での調査により，コードレビューを通して実施されたソースコード改善と不具合修正の出現回数を示す．目視で 384 件のソースコード改善履歴を分類した結果，ソースコードの改善内容 384 件中 211 件（56.0%）がコード改善であることを確認した．特に多く出現したのは変数名の変更（CHANGE_VALUE_NAME）であった．また，空白やインデントの追加削除（ADD REMOVE SPACE，CHANGE VALUE STYLE）のように既存の静的解析ツール自動的に検出可能であるにもかかわらずコードレビューにおいて改善されたのは 384 件中 46 件（11.5%）含まれていることを明らかにした．既存の静的解析ツールでは検出が困難なプロジェクトの固有名詞や出力

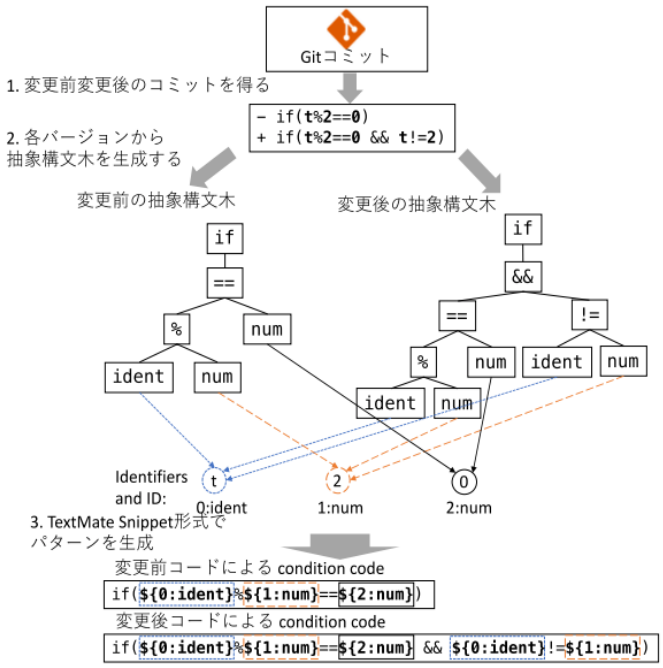

図 2　Git を用いた修正パターン生成プロセス

文字列の記法に関する修正は，各組織が独自にコーディングガイドラインを定義し，開発者間で共有することでこれらの問題を予防することが可能である．

本研究課題では，コードレビューを通して多数のコード改善が実施されていることを明らかにした [4] [5]．ソフトウェアのオープン化に伴い，不特定多数の開発者が入れ替わりながら開発を進める中で，コードレビューにおいて改善されるソースコードの変更パターンは時間経過とともに変化していることも明らかにした [6]．これら調査結果を受けて，本研究では，過去に開発者が行ったソースコードの検証，および，修正提案を行うソースコード自動検証システム DevReplay を開発した．

② DevReplay: ソースコード自動検証システム

本研究は，コードレビューを通して多数のコード改善が実施されていることを明らかにした [6] [5]．また，ソフトウェアのオープン化に伴い，不特定多数の開発者が入れ替わりながら開発を進める中で，コードレビューにおいて改善されるソースコードの変更パターンは時間経過とともに変化していることも明らかにした [4]．これら調査結果を受けて，本研究課題では，過去に開発者が行ったソースコードの検証，および，修正提案を行うソースコード自動検証システム DevReplay を開発した．DevReplay は，GitHub で公開されているソフトウェアから，ソースコードの開発履歴を収集し，他の開発者によって実装提案されたソースコードの修正方法を自分のソースコードに適用する開発環境である．DevReplay により，開発者が過去に直面したことのない問題も，関連プロジェクトから修正方法を知ることが可能となる．また，実行速度に優れたソフトウェアの開発履歴を利用することで，自身のソースコードをパフォーマンスに優れたものに改善することも可能となる．

図 2 は，DevReplay において Git を用いた場合のプログラム修正パターンの生成プロセスを示す．まず対象となるリポジトリから変更前変更後のコミットを得る．次に変更前，変更後のソースコードを抽象構文木に展開する．このとき変更前と変更後で共通している識別子や数字は $1 や${2:num}の形式に抽象化する．最後に抽象化し，識別子も含めた TextMate Snippet を修正パターンとして出力する．修正パターンに基づき，新たなソースコードの提案に対し，改善するための実装方法を提示することで自動検証を実現した．現在 DevReplay[1]は C, C++, Java, Dart, JavaScript, Python, Go, TypeScript, COBOL, Ruby, PHP, R の 12 言語に対応し，GitHub App に公開している．

（2）　大域的な改善のための保守性強化技術

本研究課題では，コードレビューにおいて検証者が判定する採択/改善要求/不採択を，従来研究で使用されたソースコードの変更規模，および本研究課題で提案する変更内容の計測結果に基づき分析した．具体的には，本研究では決定木分析により，目的変数とする検証者の判定（採択/改善要求/不採択）に最も寄与する条件で分析対象とするパッチの分類を繰り返し，検証者が判定するための条件を明らかにした［7］．

図 3 と図 4 は，jedis リポジトリを対象に，コードレビューにおいて検証者が判定する採択

[1] DevReplay: https://devreplay.github.io/

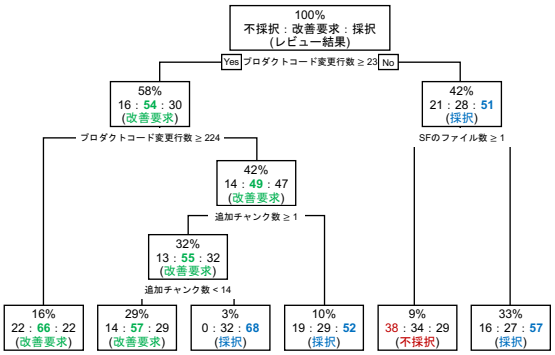

図 3 変更規模に基づく分類

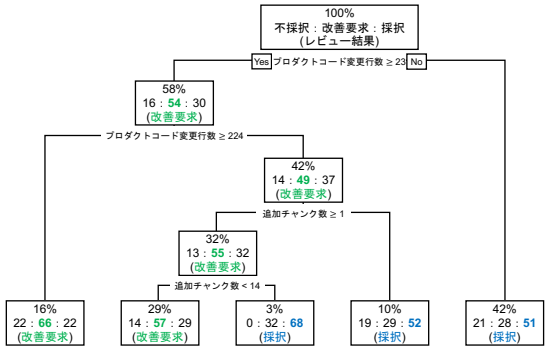

図 4 変更規模と内容に基づく分類

/改善要求/不採択を変更規模のみを用いて作成した分類木，変更規模と変更内容を用いて作成した分類木を示す． 図中の分岐点に条件と，四角枠内に分岐点の直前の条件によって分類されたプルリクエスト数の割合（枠内上部），またその判定結果の比率（枠内中部），プルリクエストの判定結果の比率が最も多い判定結果（枠内下部）を示す．分類木に基づき，検証者の判定に最も寄与する条件，および変更内容が判定に寄与するか否かについての結果をまとめる．

判定に最も寄与するメトリクス：本分析で対象とするプロジェクトでは，説明変数として変更規模に変更内容を加えたとしても，変更規模が分類に最も寄与している．具体的には図 4 の分類木の最も上位の条件において，プロダクトコード変更行数が 23 行未満の場合に，採択と判定される多数のプルリクエストを分類している．一方で，変更規模の大きいプロジェクトでは改善要求や不採択と判定されるプルリクエストを分類している．この結果から，従来研究と同様の知見を確認できた．

変更内容が判定に与える影響： ケーススタディとして対象とするプロジェクトでは，図 4 の変更規模と変更内容を用いた分類木において上位から 2 番目の条件が変更内容であることを確認できた．具体的には，パッチの 42%は，変更内容がレビュー結果に影響しており，変更行数が 23 行以下であっても，SF の（前方互換性のない変更を含む）ファイル数が 1 以上であるプルリクエストが多数不採択と判定されている．この結果から，変更内容を計測することで，後方互換性のない変更の特定を実現し，不採択または改善要求と判断されるプルリクエストを分類することができるようになった．大域的な改善のための保守性強化技術についても本研究課題で開発した DevReplay へ実装することを検討している．

参考文献

[1] Toshiki Hirao, Akinori Ihara, Kenichi Matsumoto "Pilot Study of Collective Decision-Making in the Code Review Process," In Proc. of the Center for Advanced Studies on Collaborative Research (CASCON), 2015.

[2] Yida Tao, Donggyun Han, Sunghun Kim, “Writing Acceptable Patches: An Empirical Study of Open Source Project Patches,” In Proc. of the 14th International Conference on Software Maintenance and Evolution (ICSME), 2014.

[3] Kai Pan, Sunghun Kim, and E. James Whitehead, “Toward an understanding of bug fix patterns,” Empirical Software Engineering, Vol.14, No.3, pp. 286-315, 2009.

[4] Yuki Ueda, Akinori Ihara, Takashi Ishio, Toshiki Hirao, Kenichi Matsumoto, “How are IF-Conditional Statements Fixed Through Peer Code Review,” IEICE Transactions on Information and Systems, Vol.E101.D, No.11, pp. 2720-2729, 2018.

[5] 上田裕己，石尾隆，伊原彰紀，松本健一，“コードレビューにおいて検出されるソースコード改善内容の分析,” コンピュータソフトウェア，巻 37，第 2, pp. 76-85, 2020.

[6] Yuki Ueda, Takashi Ishio, Akinori Ihara, and Kenichi Matsumoto, “Mining Source Code Improvement Patterns from Similar Code Review Works,” In Proc. of the 13th International Workshop on Software Clones (IWSC), 2019.

[7] 福元春輝，伊原彰紀，石尾隆，上田祐己，“プルリクエストにおける開発者の変更提案分類,” 情報処理学会関西支部支部大会講演論文集, 2019.

５．主な発表論文等

〔雑誌論文〕　計6件（うち査読付論文　6件／うち国際共著　1件／うちオープンアクセス　4件）

１．著者名 Yuki Ueda, Akinori Ihara, Takashi Ishio, Toshiki Hirao, Kenichi Matsumoto	４．巻 E101-D
２．論文標題 How are IF Conditional Statements Fixed Through Peer Code Review?	５．発行年 2018年
３．雑誌名 IEICE TRANSACTIONS on Information and Systems	６．最初と最後の頁 2720-2729
掲載論文のDOI（デジタルオブジェクト識別子） 10.1587/transinf.2018EDP7004	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 Shade Ruangwan, Patanamon Thongtanunam, Akinori Ihara, Kenichi Matsumoto	４．巻 24
２．論文標題 The Impact of Human Factors on the Participation Decision of Reviewers in Modern Code Review	５．発行年 2018年
３．雑誌名 Journal of Empirical Software Engineering	６．最初と最後の頁 973-1016
掲載論文のDOI（デジタルオブジェクト識別子） 10.1007/s10664-018-9646-1	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 該当する

１．著者名 Shohei Ikeda, Akinori Ihara, Raula Gaikovina Kula, Kenichi Matsumoto	４．巻 E102.D
２．論文標題 An Empirical Study of README contents for JavaScript Packages	５．発行年 2019年
３．雑誌名 IEICE Transactions on Information and Systems	６．最初と最後の頁 280-288
掲載論文のDOI（デジタルオブジェクト識別子） 10.1587/transinf.2018EDP7071	査読の有無 有
オープンアクセス オープンアクセスとしている（また、その予定である）	国際共著 －

１．著者名 Md. Rejaul Karim, Akinori Ihara, Eunjong Choi, Hajimu Iida	４．巻 31
２．論文標題 Identifying and predicting key features to support bug reporting	５．発行年 2019年
３．雑誌名 Journal of Software: Evolution and Process	６．最初と最後の頁 1-24
掲載論文のDOI（デジタルオブジェクト識別子） 10.1002/smr.2184	査読の有無 有
オープンアクセス オープンアクセスとしている（また、その予定である）	国際共著 －

1．著者名 上田 裕己，石尾 隆，伊原 彰紀，松本 健一	4．巻 37
2．論文標題 コードレビュー作業において頻繁に修正されるソースコード改善内容の分析	5．発行年 2020年
3．雑誌名 コンピュータ ソフトウェア	6．最初と最後の頁 76-85
掲載論文のDOI（デジタルオブジェクト識別子） 10.11309/jssst.37.2_76	査読の有無 有
オープンアクセス オープンアクセスとしている（また、その予定である）	国際共著 －

1．著者名 Kazumasa Shimari, Takashi Ishio, Tetsuya Kanda, Naoto Ishida, Katsuro Inoue	4．巻 206
2．論文標題 NOD4J: Near-omniscient debugging tool for Java using size-limited execution trace	5．発行年 2021年
3．雑誌名 Journal of Science of Computer Programming	6．最初と最後の頁 1-13
掲載論文のDOI（デジタルオブジェクト識別子） 10.1016/j.scico.2021.102630	査読の有無 有
オープンアクセス オープンアクセスとしている（また、その予定である）	国際共著 －

〔学会発表〕　計22件（うち招待講演　0件／うち国際学会　7件）

1．発表者名 稲垣智宏，伊原彰紀
2．発表標題 学習期間と予測期間による不具合報告数予測モデルの精度評価
3．学会等名 第26回ソフトウェア工学の基礎ワークショップ
4．発表年 2019年

1．発表者名 小口知希，伊原彰紀，稲垣智宏
2．発表標題 OSS開発者の活動量予測モデル
3．学会等名 第26回ソフトウェア工学の基礎ワークショップ
4．発表年 2019年

１．発表者名
Toshiki Hirao, Shane McIntosh, Akinori Ihara, Kenichi Matsumoto

２．発表標題
The Review Linkage Graph for Code Review Analytics

３．学会等名
The ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering 2019（国際学会）

４．発表年
2019年

１．発表者名
安東亮汰，伊原彰紀

２．発表標題
コンピューテーショナル・シンキングスキルに基づくScratchプログラムの特徴分析

３．学会等名
情報処理学会関西支部支部大会

４．発表年
2019年

１．発表者名
福元春輝，伊原彰紀

２．発表標題
プルリクエストにおける開発者の変更提案の分類

３．学会等名
情報処理学会関西支部支部大会

４．発表年
2019年

１．発表者名
才木一也，安東亮汰，伊原彰紀

２．発表標題
マイクロベンチマークサービスにおけるソフトウェアパフォーマンス改善方法の分析

３．学会等名
電子情報通信学会研究会

４．発表年
2020年

１．発表者名
上田裕己，伊原彰紀，石尾隆，桂川大輝，森田純恵，菊池慎司，松本健一

２．発表標題
ソーシャルコーディングにおけるソースコード中のIF文自動検証システムの開発

３．学会等名
マルチメディア，分散協調とモバイルシンポジウム2018

４．発表年
2018年

１．発表者名
上田裕己，伊原彰紀，石尾隆，松本健一

２．発表標題
コードレビューを通じて行われるコーディングスタイル修正の分析

３．学会等名
第25回ソフトウェア工学の基礎ワークショップ(FOSE’18)

４．発表年
2018年

１．発表者名
Yuki Ueda, Akinori Ihara, Takashi Ishio, Kenichi Matsumoto

２．発表標題
Impact of Coding Style Checker on Code Review -A case study on the OpenStack projects-

３．学会等名
The 9th International Workshop on Empirical Software Engineering in Practice (IWESEP)（国際学会）

４．発表年
2018年

１．発表者名
Yuki Ueda, Takashi Ishio, Akinori Ihara, Kenichi Matsumoto

２．発表標題
Mining Source Code Improvement Patterns from Code Review History

３．学会等名
The 13th International Workshop on Software Clones (IWSC)（国際学会）

４．発表年
2019年

１．発表者名
福元春輝，伊原彰紀

２．発表標題
コードレビューにおいて検出可能なプログラム課題の分析

３．学会等名
情報処理学会 第81回全国大会

４．発表年
2019年

１．発表者名
安東亮汰，伊原彰紀

２．発表標題
ScratchにおけるRemixが行われていない類似プログラムの特定に向けて

３．学会等名
情報処理学会 第81回全国大会

４．発表年
2019年

１．発表者名
Rodrigo Elizalde Zapata, Raula Gaikovina Kula, Bodin Chinthanet, Takashi Ishio, Kenichi Matsumoto, Akinori Ihara

２．発表標題
Towards Smoother Library Migrations: A Look at Vulnerable Dependency Migrations at Function Level for npm JavaScript Packages

３．学会等名
IEEE International Conference on Software Maintenance and Evolution (ICSME)（国際学会）

４．発表年
2018年

１．発表者名
橋谷直樹，伊原彰紀，安東亮汰

２．発表標題
Scratchにおいて再利用される作品の説明文の分析

３．学会等名
第27回ソフトウェア工学の基礎ワークショップ

４．発表年
2020年

１．発表者名
橋本大輝，伊原彰紀，小口知希

２．発表標題
社会的相互作用に着目したGitHubリポジトリへのスター付与数の見積もり手法

３．学会等名
2020年度 情報処理学会関西支部 支部大会 講演論文集

４．発表年
2020年

１．発表者名
南雄太，福元春輝，伊原彰紀

２．発表標題
コーディング規約違反解決までのソースコード特徴量の分析

３．学会等名
研究報告ソフトウェア工学（SE）

４．発表年
2021年

１．発表者名
才木一也，伊原彰紀

２．発表標題
マイクロベンチマークサービスにおけるプログラム断片の分析

３．学会等名
研究報告ソフトウェア工学（SE）

４．発表年
2021年

１．発表者名
安東亮汰，伊原彰紀

２．発表標題
Scratchプログラミング学習におけるコンピュテーショナル・シンキングスキルの習熟過程の分析

３．学会等名
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2020

４．発表年
2020年

１．発表者名
Bodin Chinthanet, Serena Elisa Ponta, Henrik Plate, Antonino Sabetta, Raula Gaikovina Kula, Takashi Ishio, Kenichi Matsumoto

２．発表標題
Code-based Vulnerability Detection in Node.js Applications: How far are we?

３．学会等名
The 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)（国際学会）

４．発表年
2020年

１．発表者名
Yuki Ueda , Takashi Ishio , Kenichi Matsumoto

２．発表標題
Automatically Customizing Static Analysis Tools to Coding Rules Really Followed by Developers

３．学会等名
The 28th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)（国際学会）

４．発表年
2021年

１．発表者名
西陽太，石尾隆，松本健一

２．発表標題
プログラミング入門科目における提出プログラムのセマンティクスを考慮した自動分類手法

３．学会等名
情報処理学会第207回ソフトウェア工学研究発表会

４．発表年
2021年

１．発表者名
Panyawut Sriiesaranusorn, Supatsara Wattanakriengkrai, Teyon Son, Takeru Tanaka, Christopher Wiraatmaja, Takashi Ishio, Raula Gaikovina Kula

２．発表標題
Kode_Stylers: Author Identification through Naturalness of Code: An Ensemble Approach

３．学会等名
Working Notes of FIRE 2020 - Forum for Information Retrieval Evaluation（国際学会）

４．発表年
2020年

〔図書〕　計0件

〔産業財産権〕

〔その他〕

―

６．研究組織

	氏名 （ローマ字氏名） （研究者番号）	所属研究機関・部局・職 （機関番号）	備考
研究分担者	石尾　隆 (Ishio Takashi) (60452413)	奈良先端科学技術大学院大学・先端科学技術研究科・准教授 (14603)	

７．科研費を使用して開催した国際研究集会

〔国際研究集会〕　計0件

８．本研究に関連して実施した国際共同研究の実施状況

共同研究相手国	相手方研究機関