ソースコードメトリクスと品質リスクとの関係分析と それに基づくリスクヘッジ手法に向けて

Risk Hedge Technique based on the Identification of Relationship between Source Code Metrics and Quality Risk

富士通株式会社 共通開発本部 プラットフォームソフトウェア開発統括部
Platform Software Development DIV., Technology development Unit, FUJITSU Ltd.
○山田　弘隆 伊藤　雅子[1)] 山瀬　清美[2)] 中嶋　久彰[3)] 長尾　徳富[4)] 小林　克己[5)]
縣　博之[6)] 森田　純恵[7)] 菊池　慎司[8)] 野中　誠[9)]
○Hirotaka Yamada　Masako Itou[1)]　Kiyomi Yamase[2)]　Hisaaki Nakajima[3)]　Tokutomi Nagao[4)]
Katsumi Hobayashi[5)]　Hiroyuki Agata[6)]　Sumie Morita[7)]　Shinji Kikuchi[8)]　Makoto Nonaka[9)]

Abstract　It has been becoming a common practice to integrate source codes developed by others (e.g. OSS) into software products. While it can reduce development cost, the black-box parts can cause various problems such as software quality degradation and prolonged root-cause analysis for failures. While source code metrics might help us evaluate software quality, most of engineers did not find it useful.
To motivate engineers to utilize software metrics, we determined the relationship between metrics representing quality degradation and quality risk parameters (e.g. times needed for modification). Through inquiries for engineers, we confirmed that our approach is valid and practical.

富士通株式会社 共通開発本部 プラットフォームソフトウェア開発統括部
Platform Software Development DIV., Technology Development Unit, FUJITSU Ltd.
〒211-8588 神奈川県川崎市中原区上小田中 4-1-1
e-mail:yamada.hirotaka@jp.fujitsu.com
1-1, Kamiodanaka 4-chome, Nakahara-ku, Kawasaki, Kanagawa 211-8588, Japan
1） 富士通株式会社　共通ソフトウェア技術本部　シニアプロフェッショナルエンジニア
Senior Professional Engineer, Software Technologies Unit, Fujitsu Ltd.
2） 富士通株式会社　プラットフォームソフトウェア事業本部
Platform Software Business Unit, Fujitsu Ltd.
3） 富士通株式会社 共通開発本部 ビジネス技術戦略室
Business Co-Creation Technology Strategy Office, Fujitsu Ltd.
4） 富士通株式会社　ミドルウェア事業本部
Middleware Business Unit, Fujitsu Ltd.
5） 富士通株式会社 共通ソフトウェア技術本部
Software Technologies Unit, Fujitsu Ltd.
6） 富士通株式会社　プラットフォームソフトウェア事業本部　シニアプロフェッショナルエンジニア
Senior Professional Engineer, Platform Software Business Unit, Fujitsu Ltd.
7） 株式会社富士通研究所　ソフトウェア研究所　主席研究員
Research Principal, Software Laboratories, FUJITSU LABORATORIES Ltd.
8） 株式会社富士通研究所　ソフトウェア研究所　主任研究員
Research Manager, Software Laboratories, FUJITSU LABORATORIES Ltd.
9） 東洋大学　経営学部　教授
Professor, Faculty of Business Administration, Toyo University

1. はじめに

近年のソフトウェア開発では、流用開発や OSS の活用など、開発者は他者が開発した機能を自身の開発対象に組み込むことが一般化してきている。このような開発方法は、類似機能の多重開発の抑止や、OSS コミュニティの開発力の活用などにより、開発工数の大幅な削減が期待できる[1]。その一方で、他者が開発した機能を取り込むことで、開発対象のソースコードがブラックボックス化・複雑化し、ソースコードの品質劣化による障害の誘発や、障害調査・修正に多くの時間を要する等の問題が発生している。開発者に開発対象のソースコードの品質情報を提供することができれば、問題の早期発見や修正時間の短縮化につながると考え、ソースコードの品質状況をメトリクスにより自動測定し、定量的に把握する仕組みの確立、および開発現場への展開を行ってきた。しかし、開発現場からは「メトリクスを見ても対応方法が判らない」等の声があがっており、定量的なメトリクス情報に基づいてソースコードの品質を改善するプロセスが現場に根付かない問題があった[2]。

そこで本稿では、ソースコードの品質劣化を表すメトリクスと、その後のソースコードの修正回数や修正延べ日数など（品質リスクと呼ぶ）との関係を分析し、これらの間に関連があることを示す。これにより、開発者に対して、メトリクスを活用したソースコード品質改善の早期対応への動機づけを行う。あわせて品質リスクへの具体的な対応案を開発現場に提示し、品質劣化を防止することで、追加コストの発生および納期遅延を未然防止することを狙いとする。

本論文の構成を以下に示す。第 2 節では、ソフトウェア品質向上へのメトリクス活用における開発現場の課題を示す。第 3 節では、品質リスクとソースコードメトリクスとの関係性の分析方法について述べる。第 4 節では、提案手法を現場に適用した結果とそれに対する考察を述べる。第 5 節で関連研究について述べた後、第 6 節にまとめと今後の課題について示す。

2. 開発現場の課題

従来、ソフトウェア開発において、各生産物の品質判断、テスト工程の完了判断、製品をリリースするための品質判断は、テスト・レビューで検出したバグの傾向分析・対策の十分性などの情報に基づいて行ってきた。この判断の基準は、過去の傾向や開発実績などの値と、開発者のそれまでの開発経験に基づいた、定性的かつ推測から導きだされたものであった。しかし近年、過去に開発されたソースコードを流用した開発（流用開発）が進み、他者が開発した機能を製品に組み込むことが多くなってきた。このような機能はブラックボックス化する要素が大きいために、経験に基づいた従来の品質判断ができない状況になっている。

流用開発においては、少ない工数で多くの機能を実装できるという利点がある反面、ソースコードの可読性の低下や接合部分のソースコードの不調和などの問題が発生しやすい。このような問題は、製品品質の劣化や障害の誘発、修正日数や修正回数の増加という品質リスクを引き起こしている。事実、筆者らの経験においては、修正回数が 3 回以上、修正日数が 30 日以上のファイルが複数存在していることを確認している。しかしこれらの問題は開発者自身が開発したソースコード以外の部分の品質に起因しているため、開発現場における対応が困難である。

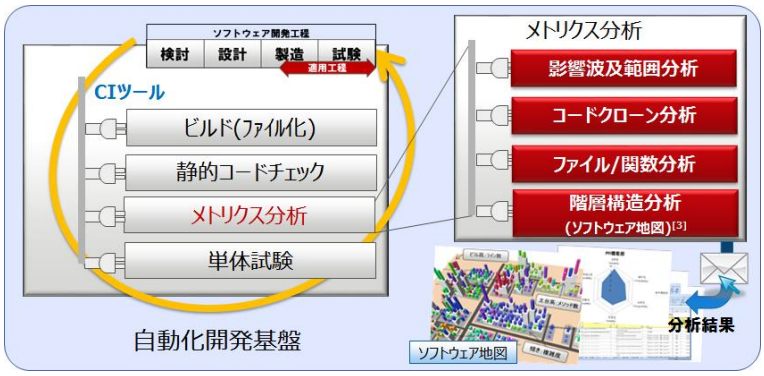


図 1　ソースコードの品質状況を定量的に把握する仕組み

そこで我々は、他者が開発したソースコード全体を把握することができなくても、それらの品質を表すメトリクスが計測・可視化できれば、ソフトウェアの品質の悪い部分を特定することができ、品質の劣化や障害修正

の長期化を抑止できると考え、ソースコードメトリクスを自動的に測定し、ソースコードの品質状況を定量的に可視化[3]する自動化開発基盤を構築した(図１)。この仕組みを現場の開発プロジェクトに適用し、開発者に対してソースコードメトリクスを示すことで、定量的指標に基づくソフトウェア品質の改善の提案を行ってきた。しかし、開発現場からは以下のような疑問の声があがっており、ソースコードメトリクスの測定結果をソースコード品質改善に活用することのメリットが認識されていないために、メトリクスを活用した品質改善のプロセスを現場で実践できていない。

［開発現場に根付かない問題］
- 関係性が不明確
  ソースコードメトリクスを見ても、どんな品質リスクに影響しているのかわからない。
- 具体的対応策が不明確
  ソースコードメトリクスを見ても、具体的な対応方法がわからない。
- 対応の効果が不明確
  ソースコードメトリクスを見てソースコード修正作業等の改善をする有効性がわからない。

これらの問題を解決するためには、以下の情報の特定が必要であり、これを本論文における課題として定義する。
- 関係性の特定：品質リスク（修正日数の増加、修正回数の増加）とソースコードメトリクスの関係を特定
- 具体的対応策の特定：ソースコードメトリクスの値に応じた具体的な品質改善策の提示
- 対応効果の提示：提示した改善策の有効性の実証

次節以降では、上記の関係性、具体的対応策、効果の特定方法について述べる。

3. 解決施策

本節では、2. で述べた課題の解決のため、品質リスクとソースコードメトリクスとの相関関係を分析する。次に、関係が見いだせたソースコードメトリクスを用いた品質リスクに対する具体的なリスクヘッジ施策を検討し、実施の有効性について評価する。

3.1 品質リスクとソースコードメトリクスとの関係性分析

以下に示す分析対象に対し、品質リスクのメトリクスとソースコードメトリクスの間の関係性を特定する。

・分析対象

開発から社内評価の期間の２製品、50 ファイル。修正が入るファイルのみ対象。

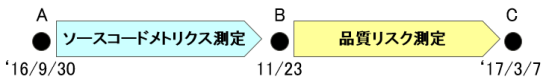


図２ メトリクス測定のタイミング

・品質リスクのメトリクス

修正日数の増加、修正回数の増加の品質リスクのメトリクスとして、ファイル単位に図２のB～C の期間で、以下の３種を測定する。これらのメトリクスは、ソース構成管理ツールのコミット履歴から自動で測定可能である。
- 修正延べ日数：問題が登録され修正完了となるまでの日数の合計。
- 修正回数：問題の修正回数。
- 平均修正日数：問題が登録され修正完了となるまでの修正日数の平均。

・ソースコードメトリクス

ソースコードメトリクスには、行数、ネスト数、コメント率などのようにモジュール単位を測定したものと、インパクトスケール（注１）やクラス結合度などのようにモジュール間の関連を測定したものがある。これらのうち、開発者にとって把握しにくいものはモジュール間の関

連に関わる性質であることから、本稿では後者に関わるメトリクスのうち、インパクトスケールを分析対象とする。また、ソースコードメトリクスの「絶対値」だけではなく、開発前段階のソースコードメトリクスがどのように変化したかに着目した「変動値」も分析対象とする。

注１）インパクトスケール[4]とはソフトウェアの一部（エンティティ）を変更した際、その影響が他のどれだけの箇所に影響するかを示す量である「影響波及量」を表すメトリクスである。図３のモデルで求める数値となっている。

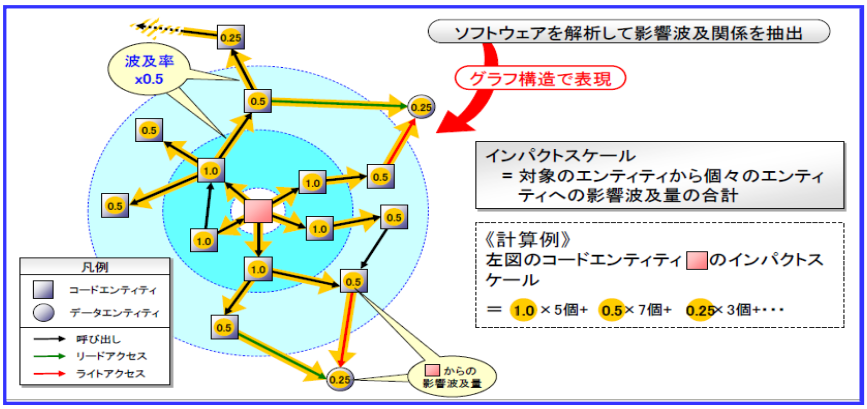


図３　インパクトスケールのモデル図

3.2 ソースコードメトリクスを用いた品質リスクのリスクヘッジ施策の検討

3.1 で関係が見いだせたソースコードメトリクスに対して、修正日数の増加、修正回数の増加といった品質リスクの回避を目的として具体的な改善策を検討する。

3.3 リスクヘッジ施策の有効性の評価

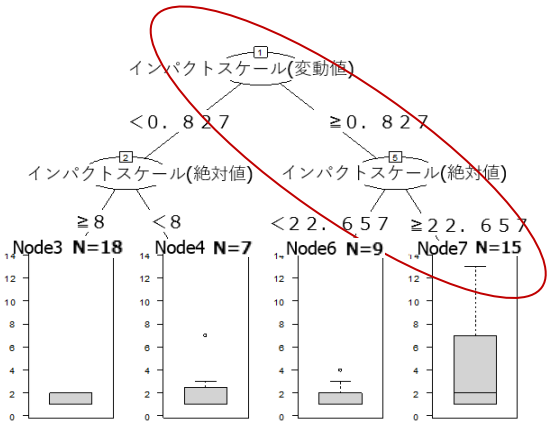
3.2 で検討したリスクヘッジ施策に対して、有効性と実現性を調査するために、ソフトウェア開発者へのアンケートを実施する。

4. 実施結果

4.1 品質リスクとソースコードメトリクスとの関係性分析結果

ソースコードメトリクスを説明変数、品質リスクのメトリクス（修正回数、修正延べ日数、平均修正日数）を目的変数として、それぞれの品質リスクについて回帰木分析を行った[5]（CARTアルゴリズムを使用）。更に、得られたセグメント（Node）間で品質リスクの値に有意な差があるかを、一元配置分散分析により検定した。

(1) 修正回数は、インパクトスケール(変動値)≧0.827 かつインパクトスケール(絶対値)≧22.657 のときに多くなる(図 4 の Node7)。
Node7 と Node3 は 0 を跨いでいないので、有意な組み合わせである(図５の Node7-3)。



	平均(mean)	標準偏差(sd)	data:n
Node3	1.277778	0.4608886	18
Node4	2.285714	2.2146697	7
Node6	1.888889	1.0540926	9
Node7	4.133333	3.8520248	15

図４　修正回数とインパクトスケールの回帰木分析

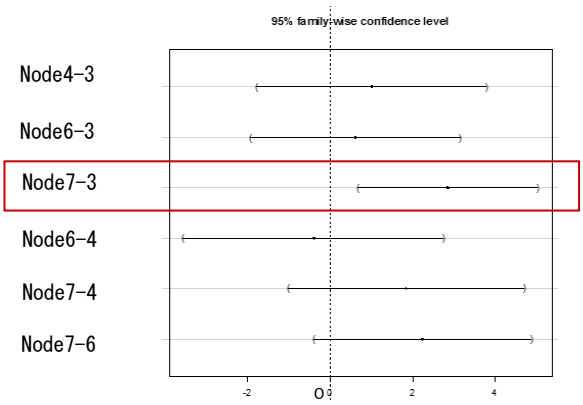
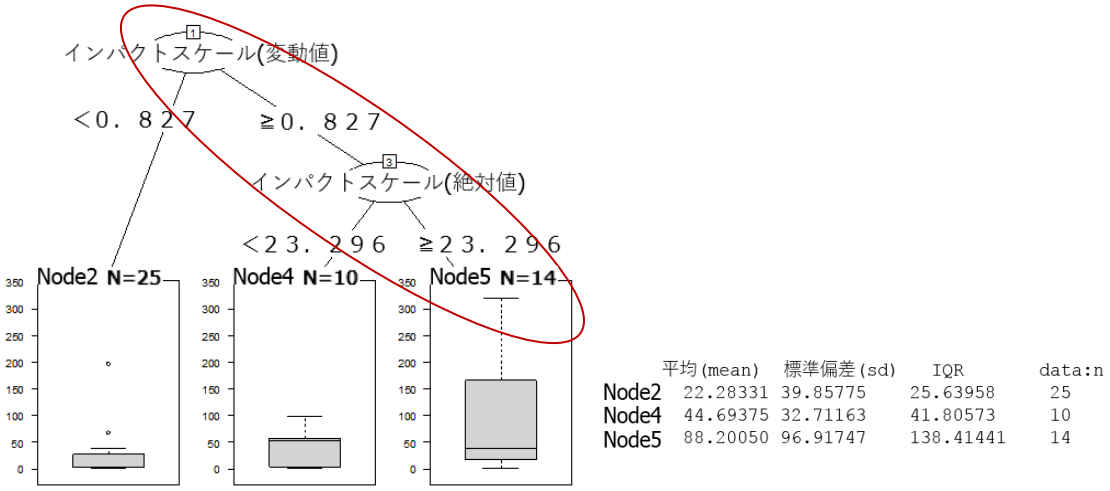


図５ 修正回数との一元配置分散分析

(2) 修正延べ日数は、インパクトスケール(変動値)≧0.827 かつインパクトスケール(絶対値)≧23.296 のときに長くなる(図６の Node5)。
Node5 と Node2 は 0 を跨いでいないので、有意な組み合わせである(図７の Node5-2)。



	平均(mean)	標準偏差(sd)	IQR	data:n
Node2	22.28331	39.85775	25.63958	25
Node4	44.69375	32.71163	41.80573	10
Node5	88.20050	96.91747	138.41441	14

図６ 修正延べ日数とインパクトスケールの回帰木分析

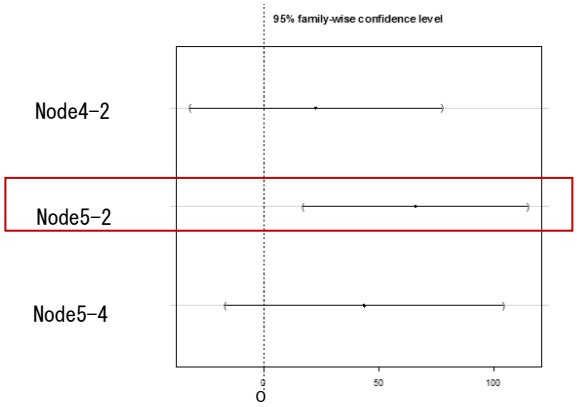


図７ 修正延べ日数との一元配置分散分析

(3) 平均修正日数についてはソースコードメトリクスとの有効な関係は見られなかった。

上記(1)と(2)で示したように、インパクトスケールが開発中に増加し一定値を超えた場合に、そのファイルで発生する修正回数や修正日数が増加するという傾向が見られた。

また、この分析結果について以下の観点で有効性の検証を行い、有効性を確認できた。これらの適合率と再現率は、文献[6]のそれと同等の水準であることなどから、一定の有効性が得られたものといえる。

- 上記(1)と(2)から求めたファイルと、開発現場の実プロジェクトが品質担保のためにとった別施策との比較検証を行った。結果として、開発現場の品質向上施策で検出した問題の修正ファイルは、メトリクスからリスクのあるファイルとして抽出したファイルにすべて含まれていた。つまり、提案手法により再現率 100%で品質問題が発生するファイルを特定できたと言える。
- (1)について「修正回数が 3 回以上のファイル」を高リスクファイルと仮定した場合の適合率、再現率を求めたところ、適合率 46%(7/15)、再現率 63%(7/11)となり、正解の網羅性を表す再現率が高かった。
- (2)について「修正延べ日数が 30 日以上のファイル」を高リスクファイルと仮定した場合の適合率、再現率を求めたところ、適合率 71%(10/14)、再現率 48%(10/21)となり、ノイズの少なさを表す適合率が高かった。

4.2 ソースコードメトリクスを用いた品質リスクのリスクヘッジ施策

4.1 の分析結果から、インパクトスケールの絶対値、変動値が一定以上を超えると修正回数、修正延べ日数が増加する品質リスクが高まることが分かった。このことからインパクトスケールに着目したリスクヘッジ施策について検討を行った。インパクトスケールの特徴からリスクヘッジ施策として検討した施策 No.1 から No.5 を表 1 に示す。

インパクトスケールの増加を抑止し修正による影響波及を小さくすることで将来の修正回数および修正延べ日数の増加を抑える方法、影響波及範囲に着目したレビューにより問題を検出する方法、インパクトスケールが大きな処理や機能について評価を重点的に実施し早期に問題を検出し手戻り工数を削減する方法を検討した。

表 1　リスクヘッジ施策一覧

No.	リスクヘッジ施策	期待効果
1	**ソースコード修正箇所・範囲を見直し** コーディング直後にインパクトスケールを測定し、増加量が一定の値を超えていた場合修正内容を見直す。	影響波及量を小さくすることで、影響範囲考慮不足による発生問題の削減、ソースコード修正時の工数の削減
2	**アーキテクチャ、データ設計の見直し** 流用開発時に母体ソースコードのインパクトスケールが大きな場合、影響波及量の小さなソフト構造に見直す。	
3	**修正影響範囲の確認コードレビュー** コーディング直後にインパクトスケールを測定し、増加量が一定の値を超えていた場合、影響波及範囲を含めてコードレビューを実施する。	修正による影響範囲を含めたコードレビューにより、影響範囲考慮不足による問題を検出する。
4	**高リスク関数の試験優先実施** インパクトスケールの絶対値、変動値が一定以上の関数/機能から試験を実施する。	早期に問題を検出することで、再試験による手戻り工数の削減を行う。
5	**高リスク関数の試験項目比重UP** 試験項目数をインパクトスケールの大きさで重み付けし、インパクトスケールが大きな関数/機能については試験項目密度(開発規模当たりの試験項目数、開発機能当たりの試験項目数)を増やす。	

4.3 リスクヘッジ施策の有効性の評価結果

4.2 で示したリスクヘッジ施策について、実プロジェクトでの有効性および実現性を評価するため、ソフトウェア開発者 37 名へのアンケートを実施した。各施策の有効性についてのアンケート結果を図８、実現性についてのアンケート結果を図９に示す。有効性についてはどの施策についても大変有効/有効と答えた開発者の合計が全体の 70%前後であり、施策として実施した場合の有効性は高いと評価する。一方で実現性については No.1、No.2 の施策に対して「できない」と答えた人数が「できる」と答えた人数と同程度存在し、これらの施策については実現性に課題があることがわかった。実施できない主な理由としては、リスクヘッジ施策として実施するには設計から見直す必要があり再試験も必要であることから実施に大きな工数が発生すること、再設計を実施するスキルを有した開発者が限られるといった理由が挙げられている。この課題を解決するために、再設計/再試験の工数を削減し、なおかつ開発者のスキルに依存しない施策の強化が必要である。この点はアジャイル型開発で用いられるプラクティスである「自動化された回帰テスト」を実施することで再試験の工数を下げることや、ソースコード解析ツールを使用してソフトウェア構造を容易に把握し再設計の難易度を下げる等、従来の開発プロセス、開発手法の革新が必要である。

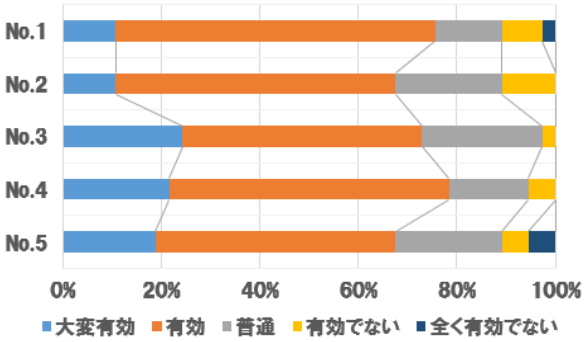


図８ 有効性に関するアンケート結果

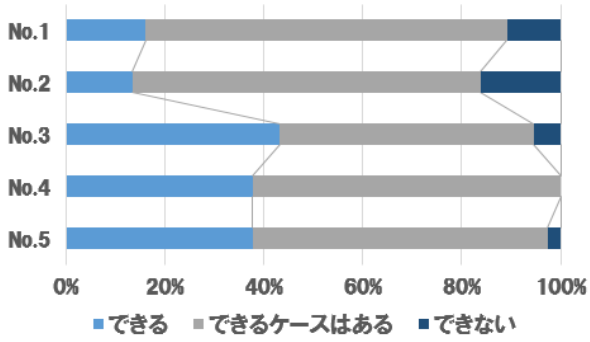


図９ 実現性に関するアンケート結果

5. 関連研究

ソフトウェア開発において、卓越したエンジニアがいればできるという時代のコード量は 100K-400K だったのに対して、近年、世界中のエンジニアが実装する OSS の世界、そして車の自動運転、繋がる世界を実現するソフトウェアのコード量は 10M/50M、100M （LoC）という膨大な量となっている。この量になると、従来のコードレビューといった属人性の高い手法が適用できなくなることからコードの静的解析ツールやテスト自動生成などの手法が主流となりつつあり、品質に関する研究も多数でている。例えば、コードメトリクスに関して、再利用に関する研究[7]、コードレビュ―のリファレンスとしての研究[8]、保守・支援に関する研究[9] があげられる。 また、アジャイル開発に関するメトリクスの研究も年々進化している[10] 。しかしながら、これらはいずれも開発計画時、コードレビュー、保守工程での活用事例であり、ひとつの開発サイクルの中でテスト工程と連携したコードメトリクスを有効に活用した品質・生産性の向上を言及しておらず、本稿の課題としてあげた豊富なコードメトリクスという有効なデータがあるにもかかわらず「開発現場に定着しない」という問題を解決するものではなかった。

6. まとめ(効果)と考察

本稿では、修正によるソースコード劣化を複数のメトリクスから求め、これと品質リスク(修正回数や修正日数)との関係性を分析した。その結果インパクトスケールが開発中に増加し一定値を超えた場合に、そのファイルで発生する修正回数や修正日数が増加することが分かった。ま

た、品質リスクへの具体的な施策を開発現場に提案し、開発者にアンケートを取ることで有効性、実現性の検証を実施した。その結果、提示した 5 種類の施策については有効性が高いと判断された一方、そのうち 2 種類(コードやデータ再設計)については、工数やスキルの観点から実現に向けての課題が大きいことも判明した。

我々の目標はソースコードメトリクスを監視することでいち早く品質リスクを検出するとともに、このリスクに対して開発者の対応方法を「リスクヘッジ手法」として明確にすることで、早期にソースコードの品質改善を行うことである。この目標に向けて今後は、品質リスクの予測をリスクヘッジが可能なタイミングで提供するために、ソースコードメトリクス測定環境で品質リスクと関連するメトリクスの監視、レポート機能を強化していく。またリスクヘッジ手法の確立に向けて施策の検討、効果測定を継続的に実施する予定である。

参考文献

[1] 森田純恵, 菊池慎司, 宗像一樹, 伊藤雅子, 土屋雅生, 小林達樹, 力武達, 野中誠. (2016). OSS 利用エンタープライズソフトウェア開発における課題調査. ソフトウェアエンジニアリングシンポジウム 2016 論文集, 2016, 2016, 112-118 (2016-08-24)

[2] 森田純恵, 菊池 慎司, 水谷 拓人, 伊藤 雅子, 土屋 雅生, 野中 誠. (2016). OSS を活用したエンタープライズソフトウェア開発向けメトリクスの抽出, ソフトウェア品質シンポジウム 2016

[3] Kobayashi, K., Kamimura, M., Yano, K., Kato, K. and Matsuo, A.:SArF Map: Visualizing software architecture from feature and layer viewpoints, Program Comprehension (ICPC), 2013 IEEE 21st International Conference on, pp. 43-52, DOI: 10.1109/ICPC.2013.6613832 (2013).

[4] 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, Feb. 2013.

[5]Terry M. Therneau, Elizabeth J. Atkinson, Mayo Foundation, An Introduction to Recursive Partitioning Using the RPART Routines, https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf(2017-07-10 参照), 2017

[6] 纐纈伸子, 川村真弥, 野村准一, 野中誠. (2010). 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. *研究報告組込みシステム (EMB)*, *2010*(6), 情報処理学会, 1-8.

[7] 鷲崎弘宜, 森田翔, 長井恭兵, 布谷貞夫, 佐藤雅宏(2012), 組込みソフトウェアの派生開発におけるソースコードメトリクスによる再利用性測定, ソフトウェア品質シンポジウム 2012

[8] 倉下亮、吉村博昭、誉田直美. 東洋大学. 野中誠 (2011), CK メトリクスの分布に基づく ソフトウェア設計の質の定量的評価, ソフトウェア品質シンポジウム 2011

[9] 小堀一雄 (2013) ソースコードの静的解析によるソフトウエア保守支援に関する研究 阪大

[10] Kammelar, J. (2015). Agile-Metrics. http://nesma.org/2015/04/agile-metrics/