コーディング規約への遵守状況を利用したソフトウェア開発プロジェクトの調査

M2021SE001 安藤勇人
指導教員：名倉正剛

1 はじめに

今日多くのオープンソースソフトウェアプロジェクト（OSS プロジェクト）が存在し，それらの OSS プロジェクトを利用し新しいソフトウェアを開発できるようになっている．その際に同じような機能を持つソフトウェアに対するプロジェクトが複数存在する場合には，なんらかの基準に基づいて利用するプロジェクトを選択する．成果物を利用するユーザは長くメインテナンスされ続けるプロジェクトの成果物を利用したいと考えているという調査結果がある [1]．また保守性の高いプロジェクトは長くメンテナンスされるという調査結果もある [2]．その場合は長くメインテナンスされ続ける OSS プロジェクトが選択される．長くメインテナンスされ続けるプロジェクトでは開発者の入替わりが多く発生したり，その結果として多くの開発者が開発に関与したりすることがある．関わる開発者が多い場合には，色々な開発者が成果物を理解する必要があり，必然的に開発成果物の保守性が高くなる．したがって，プロジェクトの開発期間や開発者数と開発成果物の保守性には何かしらの関連があると考えることができる．

そこで本研究では開発成果物の保守性を利用してプロジェクトの調査を行う．評価対象のソフトウェア開発プロジェクトに対して，そのプロジェクトの開発期間や開発者規模に対してコーディング規約をどの程度遵守しているかについて調査する．

2 研究背景と背景技術

2.1 研究背景

人気度と保守性に関連があることを導いた研究 [3] では，人気のあるプロジェクトと，人気のないプロジェクトの各種メトリクスを調査した．人気のあるプロジェクトではコードを単純化するコーディングをより多く行っていたという結果を示し，人気度とコード品質に関連があることを結論付けた．人気度と開発者数に関連があることを導いた研究 [4] では，2500 の人気のあるプロジェクトに対して，人気度と各種メトリクスの関連を調査し，開発者数と人気度に弱い相関があることを確認した．

2.2 コーディング規約

コードには読みやすさの基準というものがあり，コードは他人が最短時間で理解できるようにかかなければならない．このとき使用されるのがコーディング規約である．コーディング規約によって統一的な構文スタイルを定義することで，可読性を促進し保守性を高めることができる [5]．また，コーディング規約が読みやすさに影響を与えたという調査結果もある [6]．本研究では Google C++ Style Guide [7] のコーディング規約に準拠した静的解析ツールである Cpplint [8] を利用してコーディング規約の遵守状況を算出した．

3 コーディング規約への遵守状況とプロジェクトの開発期間や開発者数の関連の調査

3.1 研究の目的

コーディングスタイルが整っていないとソフトウェアの可読性や保守性が低下しそのことがソースコードの理解の低下につながり結果としてソフトウェアの品質を下げ，メインテナンスを困難にする．ソフトウエア開発プロジェクトにおいて関わる開発者が多いプロジェクトは，関わる開発者がソースコードを理解しやすくするために，コーディングスタイルを整えて開発を進めるはずである．また開発期間が長いプロジェクトも同様に，開発者の入れ替わりが多く発生するなど関わる開発者が必然的に多くなると考えられる．

そこで本研究ではプロジェクトの規模によってコーディング規約の遵守状況が異なると考え，プロジェクトの規模として開発者数と開発日数を定義し，開発成果物のコーディング規約の遵守状況を利用してソフトウェア開発プロジェクトを調査した．

3.2 研究課題

コーディング規約への遵守の状況が，プロジェクトの開発期間や開発者数と実際に関連があるかどうかを調査するため，次のリサーチクエスチョンを設定した．

1. プロジェクトの開発期間とコーディング規約への遵守状況に関連があるか?
2. プロジェクトの開発者数とコーディング規約への遵守状況に関連があるか?

そして多数の OSS プロジェクトを対象に観察を行うことにより，これらのリサーチクエスチョンでの問いに対する答えを明らかにする．

4 予備調査

4.1 観察対象のプロジェクト

調査対象のプロジェクトとしては，GitHub でソースコードが公開されている OSS プロジェクトのうち，主な実装言語が C++ 言語として登録されているもので，GitHub のスター数順で上位 1000 位以内のプロジェクトを利用した．それらのプロファイルを，表 1 に示す．この表において総変更行数は，追加行数と削除行数を足した値である．対象プロジェクトでは各値につき極端に大きいプロジェクトや極端に小さいプロジェクトが存在し

ていたことが，平均値と中央値の差が大きいことや，それらの値と最小値と最大値との比較からわかる．

表 1 対象プロジェクトのプロファイル

	スター数	リリース数	開発日数	実開発日数	実開発者数
最大値	166,881	5,702	15,696	9,393	93
最小値	1,805	0	1	1	1
平均値	6,024	37	2,841	995	5.7
中央値	3,300	10	2,442	481	3
	コミット数	コミッタ数	ファイル数	総変更行数	リファクタリング率
最大値	197,384	3,943	83,632	312,431,737	0.825
最小値	1	1	1	1	0
平均値	7,068	155	2,624	4,316,828	0.272
中央値	1,629	62	704	664,019	0.248

5 ソフトウェアメトリクスの選定

リサーチクエスチョン 1，2 に対して，GQM パラダイムを用いて，メインテナンスされ続けるプロジェクトの評価に使えるメトリクスを導出した．GQM パラダイムとは，[11] によって提案された，計測目的とメトリクスの対応関係が明確になり，目標と手段を整理する手法である．導出過程を図 1 に示す．長くメインテナンスされるプロジェクトは，多数の開発者が参加していること，長く開発されていること，コーディング規約に基づいた良質なコードを生成することによって，可読性や保守性が高くなると考えられる．そのため，開発者数，開発日数，リファクタリング率というメトリクスが導出された．

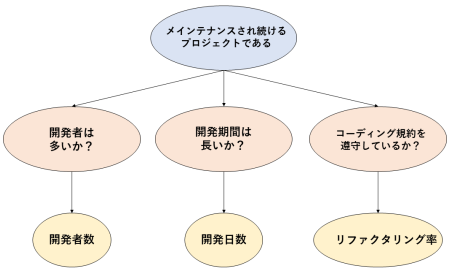

図 1 メインテナンスされ続けるプロジェクトであるかを評価することができるメトリクスの導出

5.1 観察対象のメトリクス

本研究で実施した調査では，開発期間や開発者規模に対してコーディング規約をどの程度遵守しているかを分析するために，コーディング規約違反の発生と対処の状況と開発期間や開発規模を調査した．まず違反の発生と対処の状況を表現するメトリクスと開発期間や開発規模を表現するメトリクスを定義した．

1) 実開発日数

実開発日数は，開発者がコードベースに対して少なくとも 1 つのコミットを実行した日に対応する．

2) 実開発者数

プロジェクトの中で一番コミット数が多い開発者の一割以上コミットをしている開発者を実開発者とした．

3) 違反増加数・違反減少数

名倉らの研究 [9] では，コミット単位でのコーディング規約違反の発生状況を表現する正規化コーディング規約違反メトリクス $NormV_R$ を定義している．本研究では違反の発生と違反への対処の両方を分析するために，違反増加数 $IncV_R(H)$ を (1) 式に定義し，違反減少数 $DecV_R(H)$ を (2) 式に定義する．これらは名倉らの研究 [9] で定義されている $NormV_R$ の算出式から編集量による正規化処理を引いたものである．

$$IncV_R(H) = \sum_{i=1}^{n} \phi(V_R(F_{i_aft}) - V_R(F_{i_bef})) \quad (1)$$

$$DecV_R(H) = \sum_{i=1}^{n} \phi(V_R(F_{i_bef}) - V_R(F_{i_aft})) \quad (2)$$

$$\phi(x) = \begin{cases} 0 & (\text{x<0 の場合}) \\ \text{x} & (\text{それ以外}) \end{cases}$$

これらのメトリクスは，対象のコミットでの変更に関連するファイル群 $F(F = F_1, F_2, F_3, ..., F_n)$ についてコーディング規約検査ツールによって検出される違反数の総和に対する，コミット前後での変化量である．$V_R(F_i)$ は，対象コミットでのファイル F_i のコーディング規約 R に対する規約違反箇所数であり，F_{i_bef} はファイル Fi のコミットによる変更前の状態のファイルを，F_{i_aft} は変更後の状態のファイルを示す．

さらに，プロジェクトの期間全体でのあるコーディング規約 R に対するコーディング規約違反の累積違反増加数 $IncV_R$，累積違反減少数 $DecV_R$ を，それぞれ (3) 式と (4) 式によって定義する．なおこれらは，プロジェクト内のすべてのコミットに対して算出して累積した値である．

$$IncV_R = \sum_{H} IncV_R(H) \quad (3)$$

$$DecV_R = \sum_{H} DecV_R(H) \quad (4)$$

4) リファクタリング率

RQ1，2 を解決することを目的に，発生した違反への対処状況を明らかにする目的で，リファクタリング率を定義した．長くメインテナンスされるかどうかを判断するために，リファクタリングをどの程度しているかという指標を使用する．リファクタリングにどれだけ開発の手間を割いているかを調べることによって，長くメインテナンスされるかどうかを調べる．

[10] でリファクタリングにおける保守性について，リファクタリングとは，コードの外部動作を変更せずに内部構造を改善するようにソフトウェアシステムを変更するプロセスである．これはバグが発生する可能性を最小限に抑えてコードを清掃するための方法である．本質的に，すでに書かれたコードがリファクタリングによって，コードの設計が改善される．リファクタリングを使用すると，ソフトウェア開発のプロセスのバランスが変化し，設計は最初から入念に行われるものではなく，開発中に継続的に行われることとなると述べている．

これより，リファクタリングをすることによってソフトウェアが劣化することがなくなり，リファクタリングをすることによって長くプロジェクトを保つことが出来る．また，リファクタリングを行っていないプロジェクトは書きっぱなしのプロジェクトであり，劣化することとなる．このように RQ1，2 を解決することを目的に，発生した違反への対処状況を明らかにする目的でリファクタリング率を定義した．これはある特定のコーディング規約に対して発生した違反に対して，プロジェクトの全期間内で対処したかどうかを示すメトリクスである．[10] で紹介されている大きなメソッドを複数のメソッドに分割するようなリファクタリングを実施すると，Cpplint の readability/fn size という規約が解消されることがある．このように，リファクタリングが実施されると，コーディング規約に対する違反が減少することにより，リファクタリング率を累積違反増加数と累積違反減少数を用い，(5) 式のように定義した．

$$RefRatio_R = \frac{DecV_R}{IncV_R} \tag{5}$$

5.2 RQ1,2 についての観察手順

4.1 節で述べたプロジェクトに対して，コーディング規約への遵守の状況が，プロジェクトの開発期間や開発者数と実際に関連があるかどうかを次の手順で観察した．

手順 1) 変更前と変更後のソースファイルに対し，Cpplint を利用してコーディング規約違反を検出．

手順 2) 手順 1 で検出したコーディング規約違反に対して，規約ごとに規約違反箇所数を算出．

手順 3) 手順 2 で算出した規約違反箇所数を利用し，違反増加数 $IncV_R(H)$ と違反減少数 $DecV_R(H)$ を算出．

手順 4) 手順 3 で算出した違反増加数 $IncV_R(H)$ と違反減少数 $DecV_R(H)$ より累積違反増加数と累積違反減少数を計算し，各コーディング規約のリファクタリング率を算出．

手順 5) 手順 4 で算出した各コーディング規約のリファクタリング率からプロジェクトごとのリファクタリング率を算出．

手順 6) 手順 5 で得られた各プロジェクトのリファクタリング率と GQM パラダイムによって導出されたメトリクスとの関係を調査．

6 調査結果

5.2 節で述べた手順に従って，開発者数や開発期間と，コーディング規約への遵守状況のメトリクスを取得し，その関係から相関係数を求め，表 2 に示す．

表 2 開発者数や開発期間と，リファクタリング率の相関係数

	実開発日数	開発日数	実開発者数	コミッタ数
リファクタリング率	0.58	0.51	0.24	0.38

6.1 RQ1 について

表 2 より，開発日数とリファクタリング率の関係よりも，実開発日数とリファクタリング率の関係のほうがより高い相関係数を示した．実開発日数とリファクタリング率の関係を示した散布図を，図 2 に示す．図 2 の縦軸はリファクタリング率を表し，横軸は実開発日数を表す．1 つの点が 1 プロジェクトを表しており，1000 プロジェクトが対象である．図 2 のリファクタリング率と実開発日数の分布を参照すると，実開発日数が長いプロジェクトはリファクタリング率が高く，実開発日数が短いプロジェクトのリファクタリング率は散らばりが大きい．リファクタリングはアジャイル型開発において頻繁に行われる．アジャイル型開発では，イテレーションごとに少量の成果物を生成することを重視しており，これに伴いコミット数が多くなることが予想される．しかし，実際にはコミット数が多いプロジェクトほどリファクタリング率が高くなるということは見られなかった．

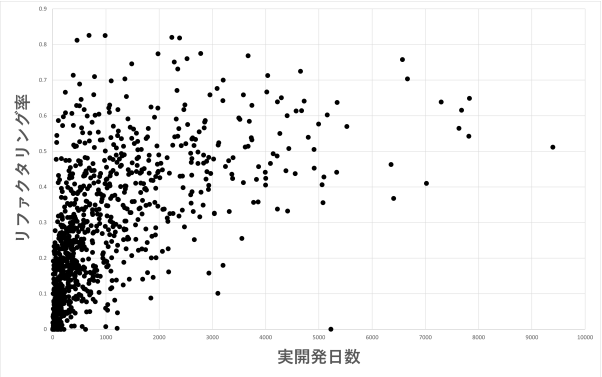

図 2 実開発日数とリファクタリング率の散布図

6.2 RQ2 について

表 2 より，実開発者数とリファクタリング率の関係よりも，コミッタ数とリファクタリング率の関係のほうがより高い相関係数を示した．コミッタ数とリファクタリング率の関係を示した散布図を，図 3 に示す．図 3 の縦軸はリファクタリング率を表し，横軸はコミッタを表す．1 つの点が 1 プロジェクトを表しており，1000 プロジェクトが対象である．図 3 のリファクタリング率とコミッタの分布を参照すると，コミッタ数が多いプロジェクトはリファクタリング率が高い．コミッタ数が少ないプロジェクトのリファクタリング率は広範囲に分布することが分かる．

6.3 プロジェクトのコミッタ数とリファクタリング率の関係の考察

結果からプロジェクトの開発者数とコーディング規約への遵守状況に弱い相関があることが分かった．図 3 から，コミッタ数が多く，リファクタリング率が低いプロジェクトは少ないことが見て取れる．またこのように多くのコミッタによって開発される場合，リファクタリング率が高くなっていた．

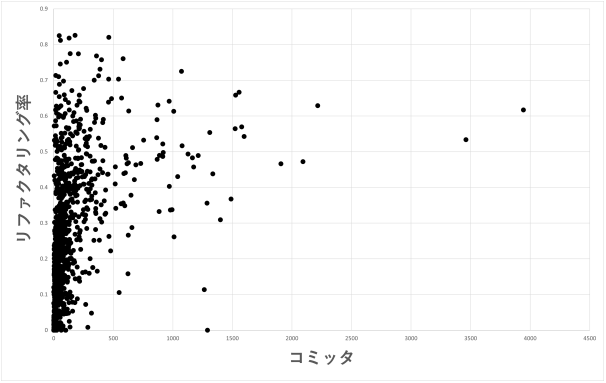

図 3 コミッタ数とリファクタリング率の散布図

そこで次に，コミッタ数が多くリファクタリング率の高いプロジェクトはどれくらいのコミッタが変更を行っているかを調査した．調査対象として，図 3 よりコミッタ数が多いプロジェクトはリファクタリング率も総じて高くなることから，コミッタ数が多いプロジェクト上位 3 件のプロジェクトを対象とした．それら上位 3 件のプロジェクトの変更回数の多いファイルに対して実際にどれくらいのコミッタが変更を行っているかを調査した．更新頻度の高いファイル上位 4 件に対して調査を行った．

コミッタ数が多いプロジェクト上位 3 件のプロジェクトのコミッタ数と変更回数の多いファイル上位 4 ファイルにおけるコミッタの比率を表 3 に示す．コミッタ数は平均値を算出し，全コミッタからみたコミッタの比率を算出した．表 3 から分かるように，コミッタ数が多いプロジェクトでもそれらの多くのコミッタによって開発されていることはなく，全コミッタの 2%未満という特定のコミッタによってのみ開発が実施されていることが分かった．また，一つのアカウントでコミットを行っているが実際にそのファイルに変更を行った開発者はそれに比べて多いことが分かった．同じアカウントを使用しているが，複数の開発者によってコードが変更されているという状況は，チームメンバーが一貫した作業を行うことが出来ること，ブランチ管理がしやすくなることといったメリットが存在する．このように一つのアカウントからコミットをすることで，チームワークを高め，プロジェクトの管理をしやすくすることで，長くメインテナンスされるプロジェクトになったのではないかと考えることができる．

表 3 各プロジェクトの変更回数が多いファイルにおけるコミッタ数の比率

プロジェクト名	上位 4 ファイルの平均コミッタ数	全コミッタのコミッタ比率
tensorflow/tensorflow	6.5	0.16%
pytorch/pytorch	6.5	0.18%
CleverRaven/Cataclysm-DDA	31.25	1.4%

7 まとめ

本研究では，プロジェクトの規模によってコーディング規約の遵守状況が異なると考え，プロジェクトの開発期間や開発者数と，コーディング規約への遵守状況に関連があるかを調査した．調査した結果，開発期間や開発者数と，リファクタリング率には相関があることを確認した．次に，開発者数とリファクタリング率に正の相関があることから，コミッタ数が多くリファクタリング率の高い 3 プロジェクトを対象に，変更回数の多いファイルに対して実際にどれくらいのコミッタが変更を行っているかを調査した．調査した結果，コミッタ数が多いプロジェクトでもそれらの多くのコミッタによって開発が実施されていることは無く，全コミッタの 2% 未満という特定のコミッタによってのみ開発が実施されていることが分かった．また，一つのアカウントでコミットを行っているにも関わらず，実際にそのファイルに変更を加えた開発者の数はそれよりも多いことを確認した．

参考文献

[1] Yamashita, K., Kamei, Y., McIntosh, S., Hassan, A. E., and Ubayashi, N.：“Magnet or Sticky? Measuring Project Characteristics from the Perspective of Developer Attraction and Retention” Journal of Information Processing 24.2：pp. 339-348(2016).

[2] Coelho, J., Valente, M. T., Milen, L., and Silva, L. L.：“Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects” Information and Software Technology 122：10624(2020).

[3] Weber, S., and Luo, J.：“What Makes an Open Source Code Popular on Git Hub?” 2014 IEEE International Conference on Data Mining Workshop：pp. 851-855(2014).

[4] Borges, H., Hora, A., and Valente, M. T.：“Understanding the Factors That Impact the Popularity of GitHub Repositories” 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)：pp. 334-344(2016).

[5] Allamanis, M., Barr, E.T., Bird, C. and Sutton, C.：“Learning Natural Coding Conventions.” Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering：pp. 281-293(2014).

[6] dos Santos, R. M., and Gerosa, M. A.：“Impacts of coding practices on readability.” In Proceedings of the 26th Conference on Program Comprehension：pp. 277-285(2018).

[7] Weinberger, B., Silverstein, C., Eitzmann, G., Mentovai, M., and Landray, T.：“Google C++ Style Guide” https://google.github.io/styleguide/cppguide.html (参照 2023-01-14).

[8] Wikimedia Foundation, Inc.：“Cpplint, Wikipedia.” https://en.wikipedia.org/wiki/Cpplint(参照 2022-08-16).

[9] 名倉正剛，田口健介, 高田眞吾：“コーディング規約違反メトリクスに基づきソフトウェア変更に対して不具合混入を予測する手法” 情報処理学会論文誌, Vol. 61, No. 4：pp. 895–907(2020).

[10] Martin Fowler, Kent Beck, John Brant, William Opdyke’s：“Refactoring: Improving the Design of Existing Code”. Addison-Wesley Professional：pp. 9, 46(1999).

[11] V.R. Basili, G. Caldiera, and H.D. Rombah：“A code generation strategy for CORBA-based Internet applications” In Proceedings First International Enterprise Distributed Object Computing Workshop：pp. 160-169(1994).