ソフトウェア保守性を評価するメトリクス間の関連分析

馬場慎太郎，吉田則裕，楠本真二，井上克郎
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
E-mail: {s-baba,n-yosida,kusumoto,inoue}@ist.osaka-u.ac.jp

概要

ソフトウェア開発の大規模化，複雑化，開発期間の短縮化に伴い保守性の高いソフトウェアを開発することの重要性が高まっている．保守性の評価としては，これまでサイクロマチック数をはじめとした複雑度メトリクスやソースコード中の同一または類似した部分を指すコードクローンに関連したメトリクスが用いられることが多かった．しかし，これらのメトリクスの評価結果がどのような関連にあるかは確認されていなかった．そこで，本研究では複数社で共同開発された中規模情報システムを対象とし，複雑度メトリクスと，コードクローンに関連したメトリクスの相関を調査した．その結果，各メトリクスの間には相関が見られず，保守性の別の側面をそれぞれ評価していることがわかった．

1　まえがき

近年，ソフトウェア開発の大規模化や複雑化，ソフトウェアの応用分野の拡大に伴って，ソフトウェア中の欠陥やそれを原因とするシステムの故障が社会に大きな影響を与えることが多くなってきた．そこで，ソフトウェアの品質の中でも特に保守性に大きな関心が寄せられている．

従来，保守性の評価にはいわゆる複雑度メトリクスが用いられてきた [9, 11]．著名なものを挙げると，McCabe のサイクロマチック数 [12] や Chidamber と Kemerer らの CK メトリクス [4] などがある．メトリクス値計測の結果，複雑度が高いソフトウェアは保守性が低いと言われている．

一方，近来保守性に影響を与える要因としてコードクローンが指摘されている [8]．コードクローンとは，ソースコード中に存在するコード片で，同形または類似したコード片が他に存在するもののことで，いわゆる重複したコードのことである．保守の際，変更を加えるコード片にコードクローンが多く存在すると手間も多くかかるため，なるべく作りこまないことが重要である．本論文ではコードクローンに関するメトリクスのことをクローンメトリクスと呼ぶが，肥後らはクローンメトリクスとして，クローン含有率（ROC）を提案している [6, 7]．このメトリクスは，直観的には対象プログラム中のコードクローンが占める割合を表す．ROC が高いと保守性が低いと指摘されており，このメトリクスの有効性の評価も行われている．

一般にソフトウェアの複雑度は評価する視点によって異なる傾向を示す．例えば，CK メトリクスでは，オブジェクト指向プログラムの複雑さを，クラスの内部，継承，結合の 3 つの視点から評価しており，それぞれが独立した特性を評価している．しかし，コードクローンに基づく評価がプログラムの保守性を評価する上で，複雑度メトリクスとどのような関連があるかは確認されていない．そこで，本研究ではこれらの保守性の評価結果において違いがあるのか確認することを目的とした．実験の対象としたのは複数社によって共同で行われたプローブ情報システム [13] の開発である．これに複雑度メトリクスとクローンメトリクスを適用し，保守性の評価結果の相関を Spearman の順位相関係数によって検定した．

2 準備

2.1 ソフトウェアの保守性

ソフトウェアの品質には様々なものがあるが，ISO/IEC9126 ではソフトウェアの保守性を次のように定義している [15].

保守性： 修正のしやすさに関する能力．修正は，是正もしくは向上，又は環境の変化，要求仕様の変更及び機能仕様の変更にソフトウェアを適応させることを含めてもよい．

一般に，ソフトウェアの保守性を評価するためには複雑度メトリクスやコードクローンに関するメトリクスが用いられることが多い．

2.2 ソフトウェアの複雑さ

ソフトウェアの保守性を評価するための方法の 1 つとして複雑さを評価する方法がある．複雑さの評価手法としては以下のようなものが挙げられる．

- サイクロマチック数

  サイクロマチック数とは，McCabe によって提案されたメトリクスである [12]．プログラム制御の流れを有向グラフで表現したときの枝の数 e，節点の数 n を用いて $e - n + 2$ で表される．この値は直観的にはソースコードの分岐の数に 1 を加えた数を表す．サイクロマチック数が多いと，テストケース（ホワイトボックステスト）を作成する手間がかかる，変更作業（保守作業）がしにくくなると指摘されている．

- Halstead のメトリクス

  Halstead のソフトウェアサイエンスというメトリクスがある [5]．これは，ソースコード中に定義されているオペランドとオペレータの種類数と出現数に基づいて規模やバグ数，開発時間を予測するメトリクスである．本メトリクスとソフトウェア保守工程における障害密度との関係の分析も行われている [16].

- CK メトリクス

  CK メトリクスとは，Chidamber と Kemerer らの提案したメトリクスである [4]．オブジェクト指向ソフトウェアに対する代表的なメトリクスであり，クラスの複雑度を静的に評価する．計測する内容毎に 6 つのメトリクスが定義されている．全てのメトリクスにおいて，計測値が大きいほうが，複雑なクラスであり好ましくないことを示す．

- インターフェイス複雑度

  インターフェイス複雑度とは，関数に関するインターフェイスの複雑度を評価するメトリクスである．値が大きいほどソフトウェア全体の複雑度も高いと言われている．M Squared Technologies の市販ツール RSM[14] では，関数中にある引数の数と”return”の数の総和でこのメトリクス値を定義している．

2.3 コードクローン

コードクローンとは，ソースコード中に存在するコード片で，同形または類似したコード片が他に存在するもののことで，いわゆる重複したコードのことである．もし，プログラムコード中にコードクローンが存在した場合には，一般的にコードの変更等が困難であると言われ，保守性低下の一因となっている．端的な例を挙げると，コードクローンの箇所にバグがあった場合，そこだけでなく，該当部と同形または類似した箇所全てを調査する必要があり，コストがかかる．

コードクローンの研究は広く行われており，多くの研究者がコードクローンの検出や分析のためのツールを作成しているが，その定義には相違がある [1, 2, 3, 8, 10]．そのうちの 1 つ，コードクローン検出ツール CCFinder では，対象ソースコードに対して字句解析を行ってトークン列を得，さらに正規化したものに対してコードクローンの検出を行う．CCFinder におけるコードクローンの定義をここで紹介する [7]．あるトークン列中に存在する 2 つの部分トークン列 α，β が等価であるとき，α と β は互いにクローンであるという．またペア α，β をクローンペアと呼ぶ．α，β それぞれを真に包含するいかなるトークン列も等価でないとき，α，β を極大クローンと呼ぶ．また，クローンの同値類をクローンセットと

呼ぶ（図 1 参照）．ソースコード中でのクローンを特にコードクローンという．CCFinder はプログラムのソースコード中に存在する極大クローンを検出する．検出されるコードクローンの最小トークン数はユーザが前もって設定できる．

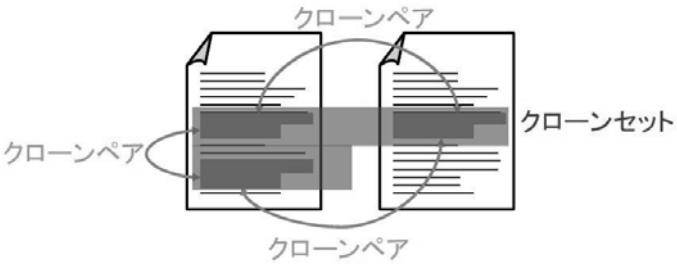

図 1. クローンペアとクローンセット

コードクローンには，いくつかの自身を特徴付けるメトリクスがある．その 1 つである RNR(S) は，クローンセット S 内に含まれるコード片がどの程度非繰り返しであるかを示す [8]．RNR が小さいコードクローンは，例えば変数宣言の羅列など，あまり保守性に影響を与えないものが多いことがわかっている．RNR は 0 から 1 までの実数値を取るが，これまでの研究で，コードクローンとしてみなすべき RNR の閾値は 0.5 程度であるという報告がされている．

また，コードクローンを用いた保守性評価のためのメトリクスとして，コードクローン含有率（ROC）がある．ROC は，その値を求める対象部分を定めれば決定され，対象部分のソースコードにおけるトークン総数を x，対象部分のソースコードにおけるトークンのうちコードクローンになっているトークンの数を y とするとき，次の式で表される．

$ROC = \frac{y}{x} \times 100(\%)$

ROC が高いほどコードクローンになっているトークンの割合が高く，保守性が低いと言われている．

3 評価実験

3.1 目的

2 節で述べたように，保守性を評価する手法としては複雑度メトリクスによるものとコードクローンによるものが著名である．しかし，これらの相関関係に関する報告はされておらず，不明である．そこで，複雑度メトリクスとクローンメトリクスによる保守性の計測結果の違いがあるかどうかを確認することを目的としている．

3.2 対象プロジェクトの概要

対象プロジェクトは，経済産業省の支援を受けて，COSE[1]参加企業によって実施されたプローブ情報システムの開発プロジェクトである．本プロジェクトは，プロジェクト管理担当の 1 社と開発担当の 5 社によって実施された．5 社のうち 2 社はそれぞれ 2 箇所の拠点で開発を行い，開発に携わった人員も異なっていたため，実質 7 社で開発したとみなすことができる．本研究では開発主体は A 社〜G 社の 7 社であるとみなす．平成 17 年度，18 年度と 2ヵ年度に渡って行われたプロジェクトで，ほとんどのプログラムは C/C++で書かれているが，一部はスクリプト言語や Python で書かれている．

3.3 分析対象データ

本研究では保守性評価のためのデータとしてはソースコードのみを用いた．対象プロジェクトは 2ヵ年度に渡って行われたが，本研究では平成 17 年度の最終段階でのデータを対象とした．また，使用ツールがサポートする言語における制約から C/C++言語で書かれている部分を対象とした．さらにプログラム中には，主にプロトタイプ宣言を行うヘッダファイルも存在するが，分析対象外とした．これには，複雑度メトリクスのうちの 1 つであるサイクロマチック数はプログラムの制御の流れを評価するにもかかわらず，ヘッダファイルには実装部がほとんどなく評価できないことや，宣言部のコードクローンは保守性にあまり影響を及ぼさない [8] という理由がある．その結果，分析対象となったプログラムは約 10 万行程度で，73 コンポーネントから構成され，ファイル数は約 400 ファイルである．

3.4 計測方法

本研究では複雑度とコードクローンという 2 つの観点から保守性を評価する．以下，それぞれの計測方法を述

[1] ソフトウェアエンジニアリング技術研究組合（COSE）:COnsortium for Software Engineering

べる．

3.4.1 複雑度

市販ツール RSM を用いて評価を行った [14]．各コンポーネントごとにサイクロマチック数とインターフェイス複雑度を計測し，それぞれを該当コンポーネントに含まれるファイルの総行数（空行とコメントのみの行は除く）で割った結果の大小で複雑度を評価している．総行数にはばらつきがあるのに対し，サイクロマチック数もインターフェイス複雑度も行数が多いほど値が大きくなるメトリクスのためこのように正規化を行った．また，サイクロマチック数とインターフェイス複雑度のみだけではなく，これらの和も複雑度を表すメトリクスとして使用した．本論文ではこれを総合複雑度と呼ぶ．なお，RSM 中でもこのメトリクスは定義され，複雑度を評価できるとされている．

3.4.2 コードクローン

コードクローン分析ツール Gemini[6] を用いて評価を行った．Gemini は内部でコードクローン検出ツール CCFinder を用いている．本研究では各コンポーネントごとに，2.3 節で説明した ROC の値を求め，その高低によって品質を評価する．

コードクローンに関しては，何をコードクローンとみなすか，ということが問題になる．コードクローンとみなす最小トークン数や，RNR というコードクローンメトリクスの閾値はツール使用時に設定できるが，今回は Gemini のデフォルト値を採用し，30 トークン以上でありかつ RNR の値が 0.5 以上のものに限りコードクローンとみなすこととした．

最小トークン数は，小さすぎるとコードクローンと呼べないような偶然に似通ったコード片を検出してしまい，大きすぎるとコードクローンと呼べるようなコード片を検出できないことが多くなるという問題がある．これまでの研究で 30 程度が妥当であると確認されている [8]．

また，本研究では会社内に閉じたコードクローンのみを対象とした．つまり，ファイルやコンポーネントが異なっても，同じ会社が開発したコード片同士ならばコードクローンとなりうるが，会社をまたがったコードクローンはありえないということである．なお，実際に会社をまたがったコードクローンは少なく，仮にコードクローンがあったとしても，各社は独立して開発しているため保守時には影響がないと思われる．

3.5 実験結果

表 1 は会社ごとのメトリクス値を表している．なお，表 1 の下に書いてあるように，各メトリクスを正規化したものを今後 Cyclo，Inter，Total と表記する．各社は 1 つ以上のコンポーネントを担当しているが，コンポーネントごとにメトリクス値を計測し，その平均値と 7 社内での順位を示している．これらのメトリクスは全て値が大きいほど保守性が低いと言われているため，メトリクス値の昇順に順位付けをしている．それぞれのメトリクスとも，会社によってある程度値が異なることがわかる．全てのメトリクスに対し，適当な 2 社を選べばその 2 社間に統計的に有意な差があることを確認した．

表 2 は Total と ROC の相関を表している．Total は，サイクロマチック数とインターフェイス複雑度両方の属性を反映しており，多角的に複雑度を評価するメトリクスだといえる．表 1 の結果を基に Spearman の順位相関係数 [17] を検定したが，相関係数は 0.11 であり，相関は確認されなかった（危険率 5 %での有意点は 0.786）．

表 3 には，他のメトリクスの組み合わせの相関を載せている．他の組み合わせでもメトリクス間に相関はないことがわかる．

表 2. Total と ROC の相関

	Total/ROC
相関係数	0.1071
有意点	0.786
相関の有無	無

3.6 分析

表 2 に示したように，それぞれ別の観点から保守性を評価している複雑度メトリクスとクローンメトリクスは，相関がないという結果になった．この結果は，それぞれ

表 1. 各社のメトリクス値

	Cyclo	順位	Inter	順位	Total	順位	ROC	順位
A 社	10.77	1	14.22	7	24.98	3	15.5	1
B 社	14.72	3	10.37	6	25.09	4	20.57	2
C 社	16.53	5	9.13	4	25.66	5	39.2	5
D 社	18.84	7	9.93	5	28.77	7	22	3
E 社	15.41	4	9.06	3	24.47	2	49.13	6
F 社	13.68	2	7.99	1	21.66	1	39	4
G 社	17.86	6	8.80	2	26.65	6	66.67	7

Cyclo：サイクロマチック数÷行数× 100
Inter：インターフェイス複雑度÷行数× 100
Total：総合複雑度÷行数× 100

表 3. メトリクス間の相関

	Cyclo/ROC	Inter/ROC	Cyclo/Inter
相関係数	0.5357	-0.7857	-0.2143
有意点	0.786	0.786	0.786
相関の有無	無	無	無

のメトリクスが持つ保守性とのギャップのベクトルが異なることを示していると思われる．よって，保守性をより正確に評価するためには，いくつかのメトリクスを組み合わせることが有用ではないかと考えられる．

また，表 3 を見ると，複雑度メトリクスとクローンメトリクスの組み合わせである Cyclo と ROC，Inter と ROC ばかりか，複雑度メトリクス同士の Cyclo と Inter の間にも相関がなかった．Inter と ROC に関しては，相関なしという結果にはなったが相関係数と有意点にはほとんど差がなく，むしろ負の相関がありそうにも思われる．再利用しないような複雑な関数の使用によって ROC が下がる，というようにトレードオフの関係があることが考えられる．

4　まとめと今後の課題

本論文では，複雑度メトリクスとクローンメトリクスによる保守性の評価結果の関連を明らかにするために，日本のある中規模情報システム開発を対象とし，ソースコードを収集し，分析を行った．その結果，複雑度メトリクスとクローンメトリクスの間には相関がないことが確認された．今後の課題としては，次のようなものが挙げられる．

- 他データへの適用

  今回，有意点と僅かな差で相関が見られなかったメトリクスの組み合わせがあった．今後他のデータも対象とし，より信頼性の高い分析結果を目指すべきである．

- メトリクスの見直し

  保守性の評価にあたって，どのようなメトリクスがふさわしいのかを改めて考える必要がある．例えば ROC は，あるトークンに対してコードクローンになっているか否かのみが評価基準であるため，多くのクローンセットの構成要素になっているトークンも，1 つだけのクローンセットの構成要素になっているトークンも同じように評価されてしまう．

- 年度間の変化

  本プロジェクトは 2ヵ年度に渡って行われた．年度間のメトリクス値変化を分析することで，ソフトウェア開発手法の変化と保守性の変化の関係を評価できると思われる．

謝辞

本研究は一部，文部科学省リーディングプロジェクト「e-Society 基盤ソフトウェアの総合開発」の委託を受けて行われた．また，一部科研費基盤研究 (A)(No.17200001) の助成を得た．

参考文献

[1] B. S. Baker. On finding duplication and near-duplication in large software systems. In *Proc. of WCRE 1995*, pp. 86–95, 1995.

[2] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis. Advanced clone-analysis to support object-oriented system refactoring. In *Proc. of WCRE 2000*, pp. 98–107, 2000.

[3] I. Baxter, A. Yahin, L. Moura, M. Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proc. of ICSM 1998*, pp. 368–377, 1998.

[4] S. R. Chidamber and C.F.Kemerer. A metrics suite for object-oriented design. *IEEE Transactions on Software Engneering*, 20(6):476–493, 1994.

[5] M. H. Halstead. Elements of software science. *Elsevier*, 1977.

[6] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue. Method and implementation for investigating code clones in a software system. *Information and Software Techonology*. in press.

[7] 肥後芳樹, 吉田則裕, 楠本真二, 井上克郎. 産学連携に基づいたコードクローン可視化手法の改良と実装. 情報処理学会論文誌, 48(2):811–822, Feb. 2007.

[8] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transaction on Software Engineering*, 28(7):654–670, Jul. 2002.

[9] 金恩美, 椿元, 楠本真二, 菊野亨. C++プログラムに対する複雑さメトリクスの提案と大学環境での実験的評価. 電子情報通信学会論文誌, J79-D-I(10):729–737, Oct. 1996.

[10] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *Proc. of SAS 2001*, pp. 40–56, 2001.

[11] 松本義弘, 肥後芳樹, 楠本真二, 井上克郎. Ck メトリクスに基づくリファクタリングの効果予測手法の提案と実装. 電子情報通信学会技術研究報告, 106(523):31–36, Feb. 2007.

[12] T. McCabe. A software complexity measure. *IEEE Transactions on Software Engneering*, SE-2:308–320, 1976.

[13] Y. Mitani, N. Kikuchi, T. Matsumura, N. Ohsugi, A. Monden, Y. Higo, K. Inoue, M. Barker, and K. Matsumoto. A proposal for analysis and prediction for software projects using collaborative filtering, in-process measurements and a benchmarks database. In *Proc. of MENSURA 2006*, pp. 98–107, 2006.

[14] RSM. `http://www.xlsoft.com/jp/products/rsm/index.html`. M Squared Technologies.

[15] JIS X 0129(ISO/IEC 9126). ソフトウェア製品の評価-品質特性及びその利用要領. 日本規格協会, 1994.

[16] 高橋良英. C 言語ソフトウェア保守工程における halstead のソフトウェアサイエンス計測と障害密度との関係の分析. 電子情報通信学会論文誌, J82-D-I(8):1017–1034, 1999.

[17] Spearman の順位相関係数. `http://kusuri-jouhou.com/statistics/soukan.html`.