Vol.2022-SE-210 No.20
2022/3/11

条件コンパイルディレクティブ中のマクロ変数によるC言語プロジェクトの複雑度の定量分析

髙津 周佑[†1,a)]　鷲崎 弘宜[†1,b)]　深澤 良彰[†1]

概要：C プリプロセッサは大規模な C 言語プロジェクトにおいて広く利用されているが，特に条件コンパイルディレクティブは主要な役割を果たしている．その反面，可読性や保守性の低下を引き起こす要因とされており，複雑な条件コンパイルディレクティブに対するリファクタリング手法への需要が高い．本論文では，条件コンパイルディレクティブの局所化によるリファクタリングを前提とし，条件中に含まれるマクロ変数に着目した上で，CV 複雑度と呼ばれるメトリクスを導入する．実験によって，このメトリクスを用いて 43 の C 言語のオープンソースプロジェクトを定量分析した．その結果，75 % のマクロ変数の利用は局所的であるものの，環境互換性目的で利用されるマクロ変数には，特に散らばっている傾向が見られることが示された．

A Quantitative Analysis of C Project Complexity by Macro Variables in Conditional Compilation Directives

Abstract: C preprocessors are widely used in large-scale C projects; especially conditional compilation directives play a major role. On the other hand, they are considered to be a source of readability and maintainability degradation, and there is a high demand for refactoring methods for complex conditional compilation directives. In this paper, we introduce a metric called CV complexity, which is based on the premise of refactoring conditional compilation directives by localizing them and focusing on the macro variables contained in the conditions. In our experiments, we quantitatively analyzed 43 C open-source projects using this metric. The results show that while the use of 75 % macro variables is localized, the macro variables used for environment compatibility purposes tend to be particularly scattered.

1. はじめに

大規模な C 言語プロジェクトにおいて，C プリプロセッサは広く利用されている．特に条件コンパイルディレクティブ（Conditional Compilation Directive; CCD）は C プリプロセッサの機能の中でも主要な機能を担う [1,2]．実際に Ernst ら [1] によると，マクロディレクティブのうち平均して 37 % が CCD を占めており，これらは機能の切り替えや環境の互換性を保つ目的で利用される．

CCD は主要な機能として広く利用されている反面，可読性や保守性を低下させる原因とされている [3–6]．これは C プリプロセッサ用の命令であるマクロディレクティブが C 言語の構文と独立しており [7]，コンパイル前の C プリプロセッサによるコード置換によって，ソースコード上の挙動と実際にコンパイルされた結果に一貫性が無いことが原因とされている [8–10]．一般に，ソフトウェアの保守作業は 40–80 %，平均して 60 % を占めるとされ [11,12]，可読性・保守性向上を目的としたリファクタリング技術確立への需要は高い．

こうした背景から，CCD に対する様々なリファクタリング方法が提案されてきた．Medeiros ら [5] は C 言語の文法中に割り込む「無規律な」CCD を，その内部のみでも完結する形にリファクタリングする方法を提示した．Adams ら [13] はアスペクト指向プログラミングを用いたリファクタリング方法を提示した．Spinellis [9,10] はブラウザで動作する C 言語プロジェクト用のリファクタリングプラットフォームを作成し，コンパイル前後の依存関係の変化をグラフによって可視化した．また，CCD 自体の整合性を検証するアルゴリズムも複数提案されている [7,14,15]．

[†1] 現在，早稲田大学
Presently with Waseda University
[a)] stakatsu@ruri.waseda.jp
[b)] washizaki@waseda.jp

本論文では，CCD に対するリファクタリングを目標として，CCD ブロックの条件中に含まれるマクロ変数（Conditional Variable; CV）に着目し，CV 複雑度（Conditional Variable Complexity; CVC）という新たなメトリクスの下で定量分析を行う．分析によって，ソフトウェア規模と CCD の関係，CCD の散らばり度合い，CCD 中のマクロ変数の特徴に応じた分類，散らばり度合いの差を明らかにした．その結果，75 % の CV は利用が局所化されているものの，散らばった CV には環境互換性用途のものが多く見られた．

本論文の貢献は（1）CV に着目した CCD の複雑度メトリクスの導入，（2）CV を用いた C 言語プロジェクトの定量分析，（3）75 % の CV は利用が局所化されている，（4）環境互換性用途の CV はより散らばって利用される傾向が強い，の四つである．

本論文の以降の構成は次の通りである．「2. 背景」では背景となる知識を導入し，動機付けの例にて複雑な CCD に対するリファクタリングを考える．「3. 提案」ではまず具体的な提案手法について説明し，次に提案の要点となる CVC の導入と CV の分類方法を提示する．「4. 評価実験」では，CVC を用いた C 言語プロジェクトの定量分析を行い，結果の分析と考察を行う．「5. 関連研究」では関連する既存の研究を提示し，本論文との関連性や比較を行う．「6. おわりに」では結論と今後の展望について述べる．

2. 背景

本章では，背景として「条件コンパイルディレクティブ」,「メソッド抽出リファクタリング」について順に説明し，最後に動機付けの例を提示する．

2.1 条件コンパイルディレクティブ

条件コンパイルディレクティブ（Conditional Compilation Directives; CCD）は C プリプロセッサのマクロディレクティブの一つで，マクロ変数の値・定義に応じた条件コンパイルを担う．C プリプロセッサとは, C コンパイラの一機能であり，コンパイル前にコード置換を行う．マクロディレクティブとは，C プリプロセッサによる具体的なコード置換方法を指定するための命令群であり，CCD の他には`#include` によるファイルインクルードや`#define` によるマクロ変数の定義等が挙げられる．

CCD の具体的な使用例としては図 1 中のソースファイル，`fun_ieee.c` や `pngread.c` に含まれる`#ifdef` ブロックが該当する．条件に含まれるマクロ変数には，開発者が独自に定義したマクロ変数や，利用するコンパイラ応じて与えられる固有の変数等がある．実際に図 1 中の`__GNUC__` は GNU コンパイラによるコンパイル時に与えられるマクロ変数である．

339–343 行目の区間を CCD ブロックと呼び，条件に応じてコンパイル前にブロック中のコードが変化する．図 1 においてはマクロ変数`__GNUC__`が定義されているかどうかに応じて変化し，定義されている場合には 340 行目のコードが，定義されていない場合には 342 行目のコードのみが，前処理完了後にそれぞれ残る．つまり，GNU コンパイラでコンパイルした場合には 340 行目の処理が採用され，それ以外のコンパイラでのコンパイル時には 342 行目の処理となる．

CCD は，マクロ変数の値に応じて実際にコンパイルされるコードの挙動が変化するため [9]，保守性を低下させる原因として知られている．また，保守性の低下に伴ってバグを多く引き起こすことも報告されている．このような現状から，複雑な CCD に対するリファクタリングの需要は大きい．

2.2 メソッド抽出リファクタリング

メソッド抽出リファクタリングとは一般的なリファクタリングパターンの一つであり [16–19]，長い処理の一部を別の新たなメソッドとして抽出することで，関心事の分離による可読性や保守性の向上が期待される．本論文では，これを C 言語の関数ではなく，CCD ブロックに対して適用することを試みる．

2.3 動機付けの例

本論文での提案の動機付けは，CCD ブロックの抽出リファクタリングである．このリファクタリング手法の狙いは，条件コンパイル部分とその他の C 言語のソースコードをファイルごと分離し，横断的関心事の局所化を図ることである．以下では，オペレーティングシステム reactos のソースコードを用いて具体的に動機付けの例を示す．図 1 は，reactos のソースファイル，`fun_ieee.c` と `pngread.c` から，条件部分に`__GNUC__`を共通して含む CCD ブロックを抽出する様子を表している．これらはいずれも`#ifdef __GNUC__`で始まる CCD ブロックを含んでおり，これらは C 言語のソースコードに混入している．

そこで，それぞれの CCD ブロックを新たな関数として抽出し，`__GNUC__.c` に移動することを考える．すると，元のソースファイルは CCD ブロックを含まない形となる．このような抽出を行うことで，関数本体の処理と CCD ブロックを別々の関心事としてファイル単位で分離することができ，可読性・保守性の向上が期待できる．

以降の章では，CCD ブロックの条件中に含まれるマクロ変数の利用の局所化という観点から複雑度のメトリクスを導入し，分析を行う．

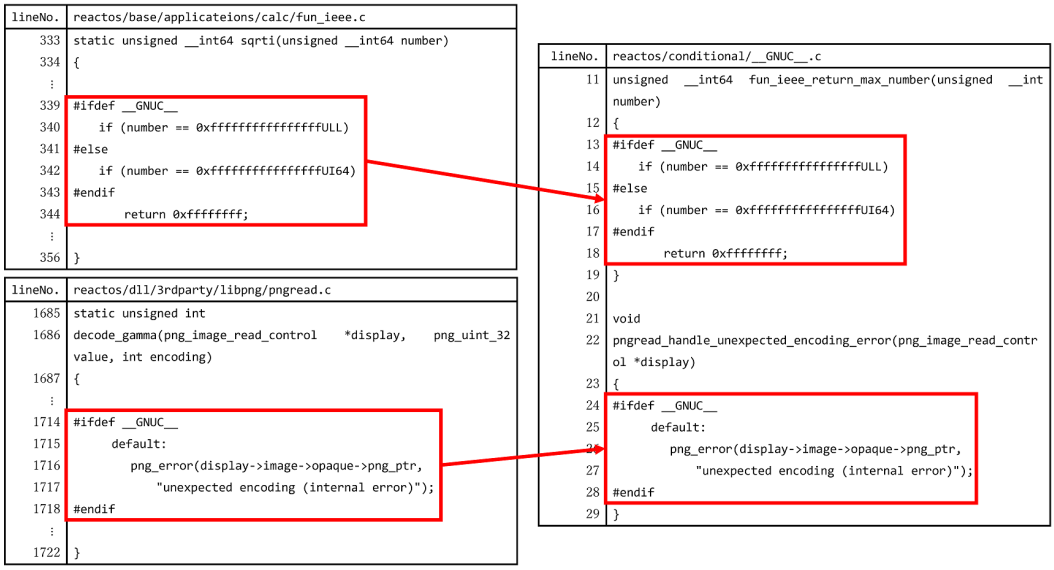

図 1　CCD ブロック抽出リファクタリングの例.
Fig. 1　Example of CCD block extraction refactoring.

3. 提案

3.1 提案の概要

複雑な CCD ブロックに対するリファクタリングを支援するため，CCD ブロックの条件に含まれるマクロ変数（Conditional Variable; CV）に着目し，CV 複雑度（Conditional Variable Complexity; CVC）というメトリクスを提案する．CVC を算出するまでの大まかな流れは，(1) マクロディレクティブの抽出，(2) ANTLR [20–22] による CCD の条件部分の構文解析，(3) `#include` 依存グラフの作成，(4) CV 依存グラフの作成，(5) CVC の算出の通りである．提案の要点は，「CVC」，「CV の種別」の二点である．以下でこの順に説明する．

3.2 CVC

CV 複雑度（Conditional Variable Complexity; CVC）は CV がどれほど多くのファイルで利用されているかを表すメトリクスである．例えば，図 2 に示す例の`__GNUC__`では，`base/applications/calc/fun_ieee.c` 中で 2 回利用されているが，CVC の算出には 1 として扱われる．CVC は小さければ小さいほど，CCD ブロック抽出リファクタリングの観点から見て理想であり，最小値は 1 である．

3.3 CV の種別

CV には CVC に応じた分類と，プロジェクト内の`#define` によるマクロ変数の定義の有無に応じた二種類の分類が定義されている．これら二つの分類は独立しており，各分類を組み合わせて sCV_{in}（分散内部定義 CV）や sCV_{ex}（分散外部定義 CV）といった CV として分類することができる．評価実験では特に，これらの sCV_{in} や sCV_{ex} に着目して議論を進める．

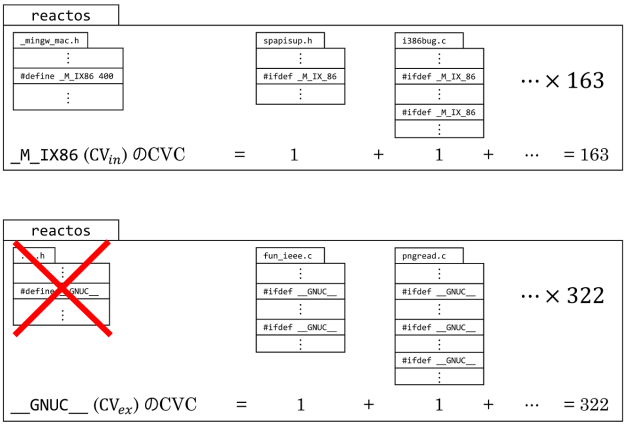

図 2　CVC の算出例（上段が CV_{in}，下段が CV_{ex} の場合）.
Fig. 2　Example of CVC calculation. (The upper row is for CV_{in}, and the lower row is for CV_{ex}.)

3.4 分散 CV

分散 CV（scattered-CV; sCV）は CVC が 2 以上の CV を表す．これは二つ以上のファイルで利用されている CV を表しており，CCD ブロック抽出リファクタリングのリファクタリング対象候補となる．

3.5 内部定義 CV と外部定義 CV

内部定義 CV（CV-internal; CV_{in}）はプロジェクト内で`#define` によって定義された CV を指す．具体的には，図 2 中の`_M_IX86` が該当し，主にコンパイル時の機能選択を切り替える目的で利用される．

外部定義 CV（CV-external; CV_{ex}）はプロジェクト内の`#define`によって定義されていない CV を表す．具体的には，図 2 中の`__GNUC__`が該当する．主に言語やコンパイラ，オペレーティングシステムの違いを反映するために与えられる．

全ての CV は CV_{in} か CV_{ex} のどちらか一方のみに属する．

4. 評価実験

4.1 実験方法

GitHub [23] より star が 10 万を超える 43 の C 言語の OSS プロジェクトを分析対象とした．以下の表 1 にその一覧を示す．尚，表中のコード行数（Lines Of Code; LOC）は空行・コメント行を削除し，行末の\が付された行は後ろの行と連結して 1 行とみなした場合の行数である．また，ソースファイル数は，ソースファイル（`.c`）とヘッダファイル（`.h`）の合計である．

以上の対象プロジェクトについて，CVC の有効性，CV の種別分布や傾向を検証するため四つの研究課題を設定する．

研究課題 1. CV に対する sCV の割合はソフトウェアの規模と共に増大するか．

研究課題 2. CV に対する sCV_{in} と sCV_{ex} の割合に違いはあるか．

研究課題 3. sCV はどの程度局所化されているか．

研究課題 4. sCV_{in} と sCV_{ex} の間に局所化度合いに違いはあるか．

研究課題 1．では，sCV の割合とソフトウェア規模の関係を調べ，規模に応じて複雑な CCD ブロックの割合が増えているかを調査する．研究課題 2．では，sCV のうち，CV_{in} と CV_{ex} に分け，プロジェクト中のマクロ変数定義の有無に応じて，sCV の割合に差異があるかどうかについて調査する．研究課題 3．では，sCV の CVC 値の分布に着目し，各プロジェクトの局所化度合いを評価する．最後に，研究課題 4．では，sCV_{in} と sCV_{ex} それぞれの分類での CVC 値の分布に着目し，局所化度合いに差があるかどうかについて評価する．

4.2 研究課題 1.

4.2.1 結果

対象プロジェクトの LOC と，CV 数に対する sCV の割合の関係について，数値は表 2 の左から 4 列に示す．

LOC と sCV の割合の相関係数は 0.16 となった．

4.2.2 考察

相関係数が 0.16 であったことから，LOC の増大に伴って

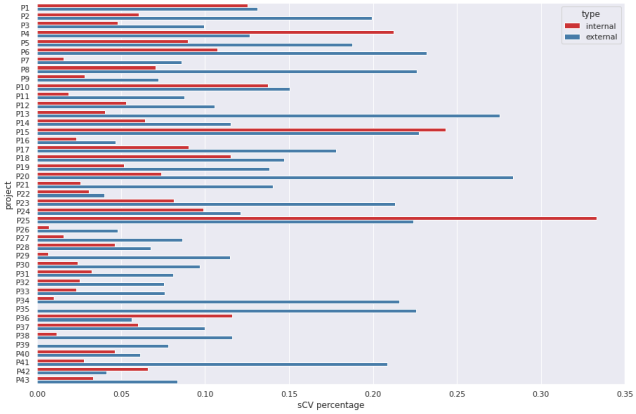

図 3 CV 数に対する sCV_{in} と sCV_{ex} の割合の比較（横軸は対数軸）．

Fig. 3 Comparison of the ratio of sCV_{in} and sCV_{ex} to the number of CVs. (Horizontal axis is the logarithmic axis.)

sCV の割合に変化は見られない．分布に着目すると，30 % を超えるプロジェクトは六つ（P4，P6，P13，P15，P20，P25）のみで，大部分は sCV の割合は 30 % 以下の範囲に収まっている．これらのプロジェクトはいずれも LOC が 50,000–100,000 の中規模なプロジェクトに該当する．考えられる原因として，中規模なプロジェクトではソースコードの規模に対するファイルの分割が十分に上手く出来ておらず，それに伴って sCV を多く含んでいる可能性がある．その証拠として，より大規模なソフトウェアでは，より sCV の割合が増大することが予想されるが，実際には sCV の割合が規模に対して小さい．大規模なソフトウェアでは設計やソースファイルの分割をすることがより求められ，その結果として sCV を一定以下に押さえることでできているのではないかと考えられる．

研究課題 1. CV に対する sCV の割合はソフトウェアの規模と共に増大するか． **sCV の割合はソフトウェア規模の増大に伴って増大しない．**

4.3 研究課題 2.

4.3.1 結果

CV 数に対する sCV_{in} と sCV_{ex} の割合を図 3 に示す．具体的な割合の数値は表 2 に示す．

結果から，43 プロジェクト中 5 プロジェクト（P4，P15，P25，P36，P42）で sCV_{in} の割合が sCV_{ex} の割合を上回り，それ以外では sCV_{ex} の割合が上回った．

4.3.2 考察

sCV_{in} と sCV_{ex} の割合の差異について，ウィルコクソンの符号順位検定を行った．帰無仮説「sCV_{in} の割合と sCV_{ex} の割合の間には差異が無い」とし，信頼区間 95 % の下で検定を行ったところ，p 値が $p < 0.01$ となり，有意な差が認められた．従って，sCV の割合については，sCV_{ex}

表 1　実験対象の C 言語プロジェクトの LOC・概要・バージョン・ディレクティブ総数・CCD 数の一覧.

Table 1　List of the LOC, summary, version, total number of directives, and number of CCD for the C language project to be tested.

ラベル	プロジェクト名	**LOC**	概要（バージョン）	ディレクティブ計	**CCD 数**
P1	reactos	5938007	オペレーティングシステム (0.4.14)	382346	114734 (30 %)
P2	php-src	1264945	プログラミング言語処理系 (8.0.14)	52656	24635 (47 %)
P3	ffmpeg	1123728	マルチメディア処理ライブラリ (n4.4.1)	50765	12732 (25 %)
P4	Tasmota	925016	Arduino 設定ファームウェア (v10.1.0)	81468	24276 (30 %)
P5	radare2	747395	リバースエンジニアリングフレームワーク (5.5.4)	46363	11471 (25 %)
P6	qmk_firmware	698267	キーボードファームウェア (0.15.13)	188974	31227 (17 %)
P7	bcc	677494	BPF (Berkeley Package Filter) 作成支援ツール (v0.23.0)	2237	378 (17 %)
P8	openssl	484297	TSL/SSL 暗号化ライブラリ (3.0.1)	40934	13579 (33 %)
P9	netdata	280362	リアルタイムパフォーマンス監視ツール (v1.32.1)	18240	10108 (55 %)
P10	micropython	279780	プログラミング言語処理系 (v1.17)	66030	20677 (31 %)
P11	git	258921	分散型バージョン管理システム (v2.34.1)	11139	3243 (29 %)
P12	obs-studio	240672	ライブストリーミング・画面録画ツール (27.1.3)	33309	13646 (41 %)
P13	tengine	227108	Web サーバ (nginx 拡張) (2.3.3)	11804	7313 (62 %)
P14	openwrt	226780	オペレーティングシステム (v21.02.1)	46877	3634 (8 %)
P15	Ventoy	225816	ブート用 USB 作成ツール (v1.0.64)	17931	5941 (33 %)
P16	mimikatz	211465	Windows 用ペネトレーションテストツール (2.2.0-20210810-2)	16478	8313 (50 %)
P17	hashcat	208598	パスワードリカバリーユーティリティ (6.2.5)	14197	5134 (36 %)
P18	curl	159412	データ転送コマンドラインツール (7.81.0)	18492	8898 (48 %)
P19	redis	153116	キー・バリューストア型データベース (6.2.6)	9153	2780 (30 %)
P20	nginx	147323	HTTP・リバースプロキシサーバ (1.21.5)	7531	4276 (57 %)
P21	mpv	141254	メディアプレイヤー (v0.34.1)	7964	1202 (15 %)
P22	timescaledb	87765	時系列 SQL 型データベース (2.5.1)	6387	1025 (16 %)
P23	zstd	85360	圧縮ライブラリ (v1.5.1)	7550	3472 (46 %)
P24	rufus	77374	USB フォーマットツール (v3.17)	7888	2588 (33 %)
P25	Nuklear	70959	クロスプラットフォーム GUI ライブラリ (4.9.5)	3470	1744 (50 %)
P26	libuv	70670	クロスプラットフォーム I/O ライブラリ (v1.43.0)	8872	4676 (53 %)
P27	tmux	59329	端末マルチプレクサ (3.2a)	1951	314 (16 %)
P28	skynet	40650	オンラインゲームフレームワーク (v1.5.0)	3283	659 (20 %)
P29	masscan	40200	TCP ポートスキャナ (1.3.2)	2269	677 (30 %)
P30	tig	35799	端末 git インターフェース (2.5.5)	1273	391 (31 %)
P31	brotli	31824	圧縮ライブラリ (v1.0.9)	1747	649 (37 %)
P32	ish	26551	iOS 向け Linux シェル (1.2.3)	3294	529 (16 %)
P33	ijkplayer	26303	Android・iOS ビデオプレイヤー (k0.8.8)	2398	666 (28 %)
P34	goaccess	25693	リアルタイムアクセスログ解析ツール (v1.5.4)	1721	459 (27 %)
P35	nginx-rtmp-module	23208	nginx 用メディアストリーミングモジュール (v1.2.2)	730	269 (37 %)
P36	jq	18885	JSON 解析コマンドラインツール (1.6)	2797	1363 (49 %)
P37	twemproxy	14806	キー・バリュー型データベースプロキシ (0.5.0)	733	131 (18 %)
P38	scrcpy	13737	デスクトップ向け Android デバイスディスプレイ (v1.21)	1215	382 (31 %)
P39	nnn	11905	端末ファイルマネージャ (v4.4)	4701	442 (9 %)
P40	wrk	4576	HTTP ベンチマークツール (4.2.0)	452	146 (32 %)
P41	the_silver_searcher	4482	文字検索コマンドラインツール (2.2.0)	575	278 (48 %)
P42	robotjs	4422	デスクトップ自動化ライブラリ (v0.6.0)	1041	512 (49 %)
P43	kcp	1490	プロトコル実装 (1.7)	207	117 (57 %)

の方がより複雑度の大きな CV を有意に多く含んでいると言える.

sCV_{in} の割合の方が大きいプロジェクトの特徴として，エンドユーザに提供する機能の割合が大きいプロジェクトが該当すると考えられる．実際に，GUI ライブラリやブート用 USB 作成ツール，設定ファームウェアといったものが該当する．この対応関係から，sCV_{in} を含む CCD ブロックはユーザ向けの機能の切り替え目的で提供されていることが予想される.

研究課題 2.　CV に対する sCV_{in} と sCV_{ex} の割合に違いはあるか．sCV_{ex} の割合の方が sCV_{in} の割合に比べて有意に大きい.

4.4　研究課題 3.

4.4.1　結果

sCV の CVC 分布を箱ひげ図に表し，CVC の第 3 四分位数と最大値を求めた．各プロジェクトの第 3 四分位数の一覧を表 2 にそれぞれ示す.

結果から，第 3 四分位数が 10 を超えるプロジェクトは，

43 プロジェクト中 2 プロジェクト（P22 と P42）であった．P22 と P42 に関しても，第 3 四分位数はそれぞれ 10.5，13.0 となっており，10 を大きく超えてはいない．最大 CVC には 2–1018 の範囲とばらつきがある．

4.4.2 考察

ほぼ全てのプロジェクトについて，sCV の 75 % は CVC が 10 以下に収まっており，CV の利用は全体的に局所化されていることが言える．また，最大 CVC は大まかにプロジェクトの規模と比例して増大する傾向が認められる．従って，75 % の sCV の CVC は 10 以下であるが，残りの 25 % ではプロジェクトの増大に伴って CVC の値に大きな差が見られる．

研究課題 3. sCV はどの程度局所化されているか． **ほぼ全てのプロジェクトで sCV は 75 % が CVC ≤ 10 の範囲に収まっており，局所化されている．**

4.5 研究課題 4.

4.5.1 結果

sCV_{in} と sCV_{ex} の CVC の分布を箱ひげ図に表した．また，「sCV_{in} と sCV_{ex} の分布の間に差がない」という帰無仮説の下，95 % 信頼区間でマン・ホイットニーの U 検定を各プロジェクトに対して行った．但し，検定にあたって十分な標本数を確保するため，sCV_{in}・sCV_{ex} の数量が共に 20 以上のプロジェクトのみを検定対象とした．各プロジェクトの検定結果と CVC の値の第 3 四分位数と最大値を併せた一覧を表 2 に示す．また，U 検定の対象となったプロジェクトのみを抜粋した箱ひげ図を図 4 に示す．

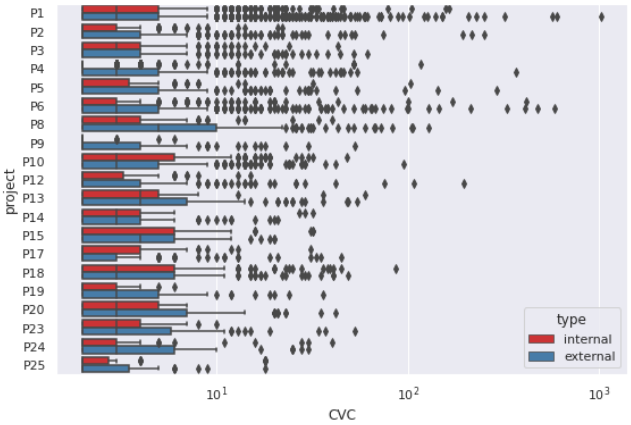

図 4 U 検定対象となった sCV_{in} と sCV_{ex} についての CVC 分布の比較（横軸は対数軸）．

Fig. 4 Comparison of the CVC distributions for sCV_{in} and sCV_{ex} subjected to the U test. (Horizontal axis is the logarithmic axis.)

結果から，43 プロジェクト中 8 プロジェクト（P11，P13，P14，P15，P18，P24，P28，P37）が sCV_{in} の最大 CVC が sCV_{ex} よりも上回っており，それ以外は全て sCV_{ex} の方が上回っていた．

4.5.2 考察

U 検定において棄却されなかったプロジェクトには，sCV_{in} に CCD ブロック中で定義されているものが多く見られた．具体的には `WIN32` や `MACOSX` 等が該当し，これらは主に sCV_{ex} でガードされた CCD ブロック中で定義され，sCV_{in} でありながら sCV_{ex} のような役割を果たしている．これらは OS 毎の違いを吸収する目的で，コンパイラ毎に異なる sCV_{ex} を一つにまとめて新たに一つの sCV_{in} として定義・利用する目的で利用されている．こういった特殊な sCV_{in} の割合や傾向・特徴を今後明らかにし，sCV_{ex} の方がより散らばって利用されている事実がより補強されることが期待される．

研究課題 4. sCV_{in} と sCV_{ex} の間に局所化度合いに違いはあるか． **多くのプロジェクトで sCV_{ex} の方がより多くのファイルで利用されている．**

4.6 妥当性への脅威

実験において，`#include` 依存グラフの生成方法が sCV_{in} と sCV_{ex} の分類時に影響した可能性があり，これは内的妥当性への脅威である．`#include` 依存グラフは，現状同名ヘッダファイルをソースファイル中から全て探索し，候補が複数見つかった場合に最もパスの近いファイルをインクルード先として決定している．今後 make ファイルの解析等も併せて行い，より実際のコンパイル時の挙動を正確に追跡できるよう拡張したい．

本実験では，これは外的妥当性への脅威である．特に 1,000,000 行を超えるプロジェクトが三つしか存在せず，大規模なプロジェクトでの結果がそのまま一般化可能とは限らない．これは外的妥当性への脅威である．今後，1,000,000 行を超えるリポジトリへの実験を拡充し，結果の有効性を検証したい．

5. 関連研究

Ernst ら [1] はマクロディレクティブ全体の使用例の概観・パターンを提示した．特に，`#define` や CCD の使用例について，パターン分類とその割合を 26 のパッケージに対して解明した．この成果は，本論文の前提となっており，特に CCD ブロックが C プリプロセッサにおけるコード置換において主要な役割を果たしている事実を支持する．

Jeffrey ら [3] は，マクロディレクティブを含む C 言語のリファクタリング手法を確立した．手法によって退行を引き起こすテストケースの存在を実証し，C 言語との依存性を考慮した上で完全なリファクタリングの提供を実現した．本論文との関連は，マクロディレクティブの構文解析部分で競合するものの，想定しているリファクタリングの方法が異なる．

表 2 sCV，sCV_{in}，sCV_{ex} の CVC 値分布と分布の差異の U 検定結果
（単位はいずれも無単位．$CVC - Q_{3/4}$ は CVC 値の第 3 四分位数を表す．
検定は CV 数が共に 20 以上のプロジェクトのみに限定）．

Table 2 sCV, sCV_{in}, sCV_{ex} distributions of CVC values and results of U-test for differences in distributions. (All units are unitless. $CVC - Q_{3/4}$ represents the third quartile of CVC values. The test is limited to only those projects for which both CVC numbers are greater than 20.)

			sCV			sCV_{in}			sCV_{ex}			U 検定	
ラベル	**LOC**	全 CV 数	CV 数	$CVC - Q_{3/4}$	最大 CVC	CV 数	$CVC - Q_{3/4}$	最大 CVC	CV 数	$CVC - Q_{3/4}$	最大 CVC	p-値	棄却
P1	5938007	13272	3404 (26 %)	5.0	1018	1663 (13 %)	5.00	163	1741 (13 %)	5.00	1018	**	o
P2	1264945	3285	854 (26 %)	3.0	252	199 (6 %)	3.00	22	655 (20 %)	4.00	252	*	o
P3	1123728	2565	378 (15 %)	4.0	61	123 (5 %)	4.00	43	255 (10 %)	4.00	61	0.9976	x
P4	925016	4132	1400 (34 %)	3.0	367	877 (21 %)	2.00	116	523 (13 %)	5.00	367	**	o
P5	747395	1462	405 (28 %)	4.0	290	131 (9 %)	3.50	103	274 (19 %)	5.00	290	*	o
P6	698267	3344	1134 (34 %)	4.0	587	359 (11 %)	3.00	415	775 (23 %)	5.00	587	**	o
P7	677494	128	13 (10 %)	4.0	9	2 (2 %)	4.50	5	11 (9 %)	4.00	9	-	-
P8	484297	1208	358 (30 %)	8.0	128	85 (7 %)	4.00	40	273 (23 %)	10.00	128	**	o
P9	280362	995	100 (10 %)	4.0	53	28 (3 %)	2.00	6	72 (7 %)	4.00	53	**	o
P10	279780	2808	808 (29 %)	5.0	95	386 (14 %)	6.00	48	422 (15 %)	5.00	95	0.9586	x
P11	258921	810	86 (11 %)	4.0	18	15 (2 %)	2.00	18	71 (9 %)	4.00	17	-	-
P12	240672	2347	372 (16 %)	4.0	195	124 (5 %)	3.25	15	248 (11 %)	4.00	195	*	o
P13	227108	592	187 (32 %)	7.0	60	24 (4 %)	5.00	60	163 (28 %)	7.00	54	0.299	x
P14	226780	686	123 (18 %)	4.0	32	44 (6 %)	4.00	32	79 (12 %)	4.00	21	0.3376	x
P15	225816	576	271 (47 %)	6.0	32	140 (24 %)	6.00	32	131 (23 %)	6.00	20	*	o
P16	211465	725	51 (7 %)	2.5	54	17 (2 %)	2.00	2	34 (5 %)	4.75	54	-	-
P17	208598	831	223 (27 %)	3.5	45	75 (9 %)	4.00	31	148 (18 %)	3.00	45	0.7209	x
P18	159412	1326	348 (26 %)	6.0	86	153 (12 %)	6.00	86	195 (15 %)	6.00	49	0.2075	x
P19	153116	636	121 (19 %)	4.0	36	33 (5 %)	3.00	6	88 (14 %)	5.00	36	**	o
P20	147323	406	145 (36 %)	6.0	42	30 (7 %)	5.00	7	115 (28 %)	7.00	42	0.082	x
P21	141254	349	58 (17 %)	4.0	11	9 (3 %)	3.00	6	49 (14 %)	4.00	11	-	-
P22	87765	226	16 (7 %)	10.5	28	7 (3 %)	11.00	17	9 (4 %)	8.00	28	-	-
P23	85360	554	163 (29 %)	5.0	53	45 (8 %)	4.00	10	118 (21 %)	5.75	53	0.1284	x
P24	77374	496	109 (22 %)	5.0	40	49 (10 %)	3.00	40	60 (12 %)	6.00	30	**	o
P25	70959	210	117 (56 %)	3.0	18	70 (33 %)	2.75	18	47 (22 %)	3.50	18	*	o
P26	70670	1419	78 (5 %)	7.0	72	10 (1 %)	3.00	5	68 (5 %)	7.25	72	-	-
P27	59329	127	13 (10 %)	2.0	4	2 (2 %)	3.50	4	11 (9 %)	2.00	4	-	-
P28	40650	237	27 (11 %)	3.0	5	11 (5 %)	3.50	5	16 (7 %)	3.00	4	-	-
P29	40200	157	19 (12 %)	9.5	27	1 (1 %)	4.00	4	18 (11 %)	9.75	27	-	-
P30	35799	124	15 (12 %)	3.0	7	3 (2 %)	4.00	4	12 (10 %)	2.25	7	-	-
P31	31824	185	21 (11 %)	3.0	61	6 (3 %)	2.00	2	15 (8 %)	3.00	61	-	-
P32	26551	119	12 (10 %)	4.5	11	3 (3 %)	2.00	2	9 (8 %)	6.00	11	-	-
P33	26303	171	17 (10 %)	4.0	12	4 (2 %)	3.25	4	13 (8 %)	4.00	12	-	-
P34	25693	102	23 (23 %)	3.0	22	1 (1 %)	2.00	2	22 (22 %)	3.00	22	-	-
P35	23208	31	7 (23 %)	8.5	10	0 (0 %)	0.00	0	7 (23 %)	8.50	10	-	-
P36	18885	267	46 (17 %)	2.0	9	31 (12 %)	2.00	2	15 (6 %)	3.50	9	-	-
P37	14806	50	8 (16 %)	3.0	3	3 (6 %)	3.00	3	5 (10 %)	2.00	2	-	-
P38	13737	86	11 (13 %)	6.0	7	1 (1 %)	2.00	2	10 (12 %)	6.50	7	-	-
P39	11905	64	5 (8 %)	2.0	2	0 (0 %)	0.00	0	5 (8 %)	2.00	2	-	-
P40	4576	65	7 (11 %)	2.5	4	3 (5 %)	2.00	2	4 (6 %)	3.25	4	-	-
P41	4482	72	17 (24 %)	3.0	10	2 (3 %)	3.00	3	15 (21 %)	2.00	10	-	-
P42	4422	121	13 (11 %)	13.0	14	8 (7 %)	11.50	14	5 (4 %)	13.00	14	-	-
P43	1490	60	7 (12 %)	2.0	2	2 (3 %)	2.00	2	5 (8 %)	2.00	2	-	-

Medeiros ら [5] は，不完全な C 言語ブロックを含む CCD に対するリファクタリングカタログを作成し，OSS プロジェクトにパッチを送り，62 % のパッチが実際に受理された．この手法でのリファクタリング方法は単一のファイル内の CCD ブロックを対象に，その内部や周辺に存在する C 言語コードを修正しており，CCD ブロックそのものの別ファイルへの移動という点で本論文と異なる．

6. おわりに

本論文では，CVC というメトリクスを新たに導入し，CV に着目した C 言語プロジェクトの CCD の複雑度を定量分析した．その結果，複数ファイルに散らばった CV の割合はソフトウェア規模の増大とは関係が薄く，75 % の CV は 10 以下のファイルで利用され，大部分の CV の利用は局所化されている．環境互換性目的で利用される CV（CV_{ex}）の方が，複数ファイルで利用されている割合も，利用されるファイル数も大きい．これらの分析結果は，複雑になりがちな CCD の傾向を示すだけでなく，リファクタリングすべき複雑な CCD の自動的な検出指標として役立つことが期待される．

今後の展望としては，CVC が大きな複雑な CCD ブロックとバグの発生頻度の関係性について解明したい．Muniz ら [24] によって既に CCD ブロックとバグの関係は報告されているが，この成果を拡張する形で，CVC の高さとバグの関連性という観点で，定量的な評価で有効性を示したい．

参考文献

[1] Ernst, M., Badros, G. and Notkin, D.: An empirical analysis of c preprocessor use, IEEE Transactions on Software Engineering, Vol. 28, No. 12, pp. 1146–1170 (online), DOI: 10.1109/TSE.2002.1158288 (2002).

[2] Kästner, C., Giarrusso, P. G., Rendel, T., Erdweg, S., Ostermann, K. and Berger, T.: Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation, SIGPLAN Not., Vol. 46, No. 10, p. 805–824 (online), DOI: 10.1145/2076021.2048128 (2011).

[3] Overbey, J. L., Behrang, F. and Hafiz, M.: A Foundation for Refactoring C with Macros, Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, New York, NY, USA, Association for Computing Machinery, p. 75–85 (online), DOI: 10.1145/2635868.2635908 (2014).

[4] Liebig, J., Apel, S., Lengauer, C., Kästner, C. and Schulze, M.: An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines, Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE '10, New York, NY, USA, Association for Computing Machinery, p. 105–114 (online), DOI: 10.1145/1806799.1806819 (2010).

[5] Medeiros, F., Ribeiro, M., Gheyi, R., Apel, S., Kästner, C., Ferreira, B., Carvalho, L. and Fonseca, B.: Discipline Matters: Refactoring of Preprocessor Directives in the #ifdef Hell, IEEE Transactions on Software Engineering, Vol. 44, No. 5, pp. 453–469 (online), DOI: 10.1109/TSE.2017.2688333 (2018).

[6] Spencer, H. and Collyer, G.: #ifdef Considered Harmful, or Portability Experience with C News, USENIX Summer 1992 Technical Conference (USENIX Summer 1992 Technical Conference), San Antonio, TX, USENIX Association, (online), available from ⟨https://www.usenix.org/conference/usenix-summer-1992-technical-conference/ifdef-considered-harmful-or-portability⟩ (1992).

[7] Badros, G. J. and Notkin, D.: A framework for preprocessor-aware C source code analyses, Software: Practice and Experience, Vol. 30, No. 8, pp. 907–924 (online), DOI: https://doi.org/10.1002/(SICI)1097-024X(20000710)30:8¡907::AID-SPE324¿3.0.CO;2-I (2000).

[8] Vidacs, L., Beszedes, A. and Ferenc, R.: Columbus schema for C/C++ preprocessing, Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings., pp. 75–84 (online), DOI: 10.1109/CSMR.2004.1281408 (2004).

[9] Spinellis, D.: CScout: A refactoring browser for C, Science of Computer Programming, Vol. 75, No. 4, pp. 216–231 (online), DOI: https://doi.org/10.1016/j.scico.2009.09.003 (2010).

[10] Spinellis, D.: Global analysis and transformations in preprocessed languages, IEEE Transactions on Software Engineering, Vol. 29, No. 11, pp. 1019–1030 (online), DOI: 10.1109/TSE.2003.1245303 (2003).

[11] Glass, R.: Frequently forgotten fundamental facts about software engineering, IEEE Software, Vol. 18, No. 3, pp. 112–111 (online), DOI: 10.1109/MS.2001.922739 (2001).

[12] Tiwari, D., Washizaki, H., Fukazawa, Y., Fukuoka, T., Tamaki, J., Hosotani, N. and Kohama, M.: Metrics Driven Architectural Analysis using Dependency Graphs for C Language Projects, 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Vol. 1, pp. 117–122 (online), DOI: 10.1109/COMPSAC.2019.00025 (2019).

[13] Adams, B., De Meuter, W., Tromp, H. and Hassan, A. E.: Can We Refactor Conditional Compilation into Aspects?, Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development, AOSD '09, New York, NY, USA, Association for Computing Machinery, p. 243–254 (online), DOI: 10.1145/1509239.1509274 (2009).

[14] Saebjoernsen, A., Jiang, L., Quinlan, D. and Su, Z.: Static Validation of C Preprocessor Macros, 2009 IEEE/ACM International Conference on Automated Software Engineering, pp. 149–160 (online), DOI: 10.1109/ASE.2009.75 (2009).

[15] Garrido, A. and Johnson, R.: Analyzing multiple configurations of a C program, 21st IEEE International Conference on Software Maintenance (ICSM'05), pp. 379–388 (online), DOI: 10.1109/ICSM.2005.23 (2005).

[16] Fowler, M.: Refactoring: Improving the Design of Existing Code, Extreme Programming and Agile Methods — XP/Agile Universe 2002 (Wells, D. and Williams, L., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 256–256 (2002).

[17] Murphy-Hill, E., Parnin, C. and Black, A. P.: How we refactor, and how we know it, 2009 IEEE 31st International Conference on Software Engineering, pp. 287–297 (online), DOI: 10.1109/ICSE.2009.5070529 (2009).

[18] 後藤　祥，吉田則裕，藤原賢二，崔　恩瀞，井上克郎：メソッド抽出リファクタリングが行われるメソッドの特徴調査，コンピュータソフトウェア，Vol. 31, No. 3, pp. 3_318–3_324（オンライン），DOI: 10.11309/jssst.31.3_318 (2014).

[19] 後藤　祥，吉田則裕，藤原賢二，崔　恩瀞，井上克郎：機械学習を用いたメソッド抽出リファクタリングの推薦手法，情報処理学会論文誌，Vol. 56, No. 2, pp. 627–636（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/110009877377/⟩ (2015).

[20] : ANTLR, https://www.antlr.org/.

[21] Parr, T. J. and Quong, R. W.: ANTLR: A predicated-LL(k) parser generator, Software: Practice and Experience, Vol. 25, No. 7, pp. 789–810 (online), DOI: https://doi.org/10.1002/spe.4380250705 (1995).

[22] Parr, T. and Fisher, K.: LL(*): The Foundation of the ANTLR Parser Generator, SIGPLAN Not., Vol. 46, No. 6, p. 425–436 (online), DOI: 10.1145/1993316.1993548 (2011).

[23] : GitHub: Where the world builds software · GitHub, https://github.com.

[24] Muniz, R., Braz, L., Gheyi, R., Andrade, W., Fonseca, B. and Ribeiro, M.: A Qualitative Analysis of Variability Weaknesses in Configurable Systems with #ifdefs, Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems, VAMOS 2018, New York, NY, USA, Association for Computing Machinery, p. 51–58 (online), DOI: 10.1145/3168365.3168382 (2018).