特集論文

静的解析技術による ソースコードの効率的な品質確保

藤本卓也*
中村勝彦*
浅川忠隆*

Efficient Quality Assurance of Source Code by Static Analysis Technologies

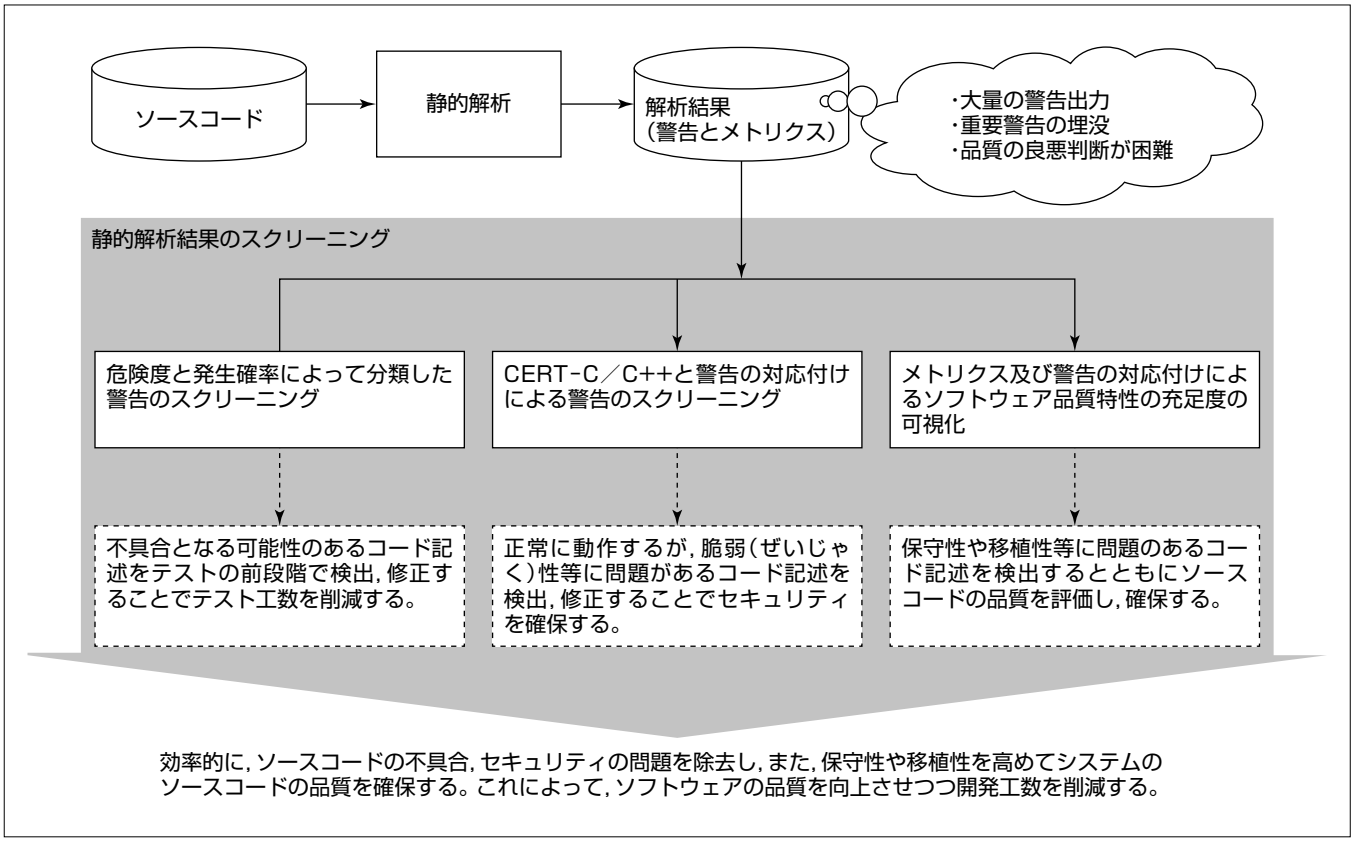
Takanari Fujimoto, Katsuhiko Nakamura, Tadataka Asakawa

要　旨

IoT(Internet of Things)機器の高機能化やネットワークの多様化によって搭載ソフトウェアの大規模化が進んでいる。これに対し，ソフトウェアのコーディング段階で，プログラムを実行せずにソースコードを機械的に解析する静的解析技術を活用し，内在する不具合となる可能性がある箇所(以下“問題箇所”という。)の検出作業を効率化し，フロントローディング率向上を図ってきた。

しかし，静的解析は警告の誤検出が多く，修正が必要な問題箇所の見極め作業に時間を要する傾向があり，大規模なソースコードに対しては誤検出による問題箇所の検出効率の悪化への対策が重要となっている。また，保守性やセキュリティ等のソフトウェア品質特性に関連付けたソースコード品質評価等，新たな観点による検証が求められており，静的解析結果から問題箇所を見極める作業の更なる効率化が必要である。

これら新たな課題への対策として，膨大な静的解析結果から問題箇所を選別するスクリーニング環境を構築して適用を図り，プログラム実行停止等につながる重大な問題箇所の絞り込み，静的解析結果の警告とメトリクスを組み合わせたソフトウェア品質特性への充足度の評価を可能とし，ソースコードの効率的な品質確保を実現した。



効率的にソースコードの品質を確保するための静的解析技術の適用

静的解析技術はソースコードの品質を確保するために有効であるが，警告が大量に出力されるため，不具合となる可能性のあるコード記述，脆弱性等に問題があるコード記述をいかに的確かつ効率的に検出するかが重要なカギとなる。また，静的解析技術によって測定されるメトリクスをコードの保守性や移植性を維持するための尺度として使用できるが，的確な判断を行うにはどのメトリクスを使い，その値をどう評価するかが重要なカギとなる。

*設計システム技術センター

1. まえがき

IoT機器の高機能化やネットワークの多様化によって，搭載されるソフトウェアの大規模化が進展している(1)。これに対して従来，ソフトウェアのコーディング段階で，プログラムを実行せずにソースコード(以下“コード”という。)を機械的に解析する静的解析技術を活用し，コードの問題箇所の検出作業を効率化し，フロントローディング率の向上を図ってきた(2)。

しかし，静的解析ツールは短時間に解析結果を得られる反面，プログラム実行を伴わない分，誤検出が多発し，真に修正が必要な問題箇所の見極め作業に時間を要する傾向がある。

このため，オープンソースソフトウェア(Open Source Software：OSS)活用の進展や多人数による開発によって大規模化が進むコードに対して，誤検出による問題箇所の検出効率の悪化への対策が重要となっている。加えて，保守性やセキュリティ等のソフトウェア品質特性に関連付けたコードの品質評価等，新たな観点による検証が求められており，静的解析結果から問題箇所を見極める作業の更なる効率化が必要である。

これら新たな課題への対策として，膨大な静的解析結果から問題箇所を選別するスクリーニング環境を構築して適用を図り，次に示すコード品質の効率的な確保を実現した。

(1) コードの大規模化に対応した，プログラム実行停止等につながる重大な問題箇所の絞り込み作業の効率化
(2) 静的解析結果からの警告とメトリクスを組み合わせたソフトウェア品質特性への充足度評価

2. 静的解析結果のスクリーニング環境

開発ライフサイクルの早期から効率的に実装品質を確保するため，静的解析結果である問題箇所を指摘する警告や複雑度等のコード品質を示す指標値であるメトリクスから，問題箇所や，ソフトウェア品質特性を悪化させるコード記述を示す警告とメトリクスを選別するスクリーニング環境(以下“スクリーニング環境”という。)を構築し，活用を図っている(**図１**)。

スクリーニング環境は２種類のデータを入力とする。１つは解析対象コードに対して静的解析が出力する大量の解析結果(以下“解析結果”という。)，もう１つはスクリーニング用設定データ(以下“設定データ”という。)である。

解析結果は，警告とメトリクスの２種のデータから成る。設定データは，解析結果から抽出すべきプログラム実行停止等につながる重大な問題箇所となる度合い(以下“危険度”という。)に基づき分類した警告リスト，ソフトウェア品質特性評価用の警告リスト及びメトリクスと推奨値リストから成る。

スクリーニング環境は解析結果と設定データとを照合し，設定データの情報に基づき，解析結果から不具合の可能性が高いコード記述や，ソフトウェア品質特性を悪化させるコード記述やメトリクスを抽出して出力する。

ソフトウェア開発者は，スクリーニング環境が出力した解析結果とコードを突き合わせることで問題箇所やソフトウェア品質特性への充足度を効率的に確認可能となる。

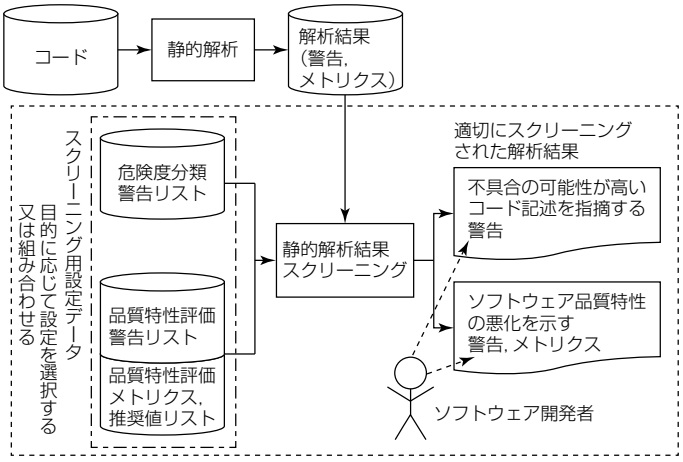


図１．静的解析結果のスクリーニング環境の構成

3. コード品質の向上を効率化するための対策

3.1 内在するコード問題点の効率的かつ的確な検出

静的解析技術は，保守性の悪化，プログラム実行停止等の問題を引き起こすコード記述を示す警告の出力やコードの品質を示すメトリクスの測定を行い，問題箇所の検出を支援する有用な技術である。しかし，問題箇所を広く検出する解析方針から解析結果には多くの誤検出が内在する。近年のIoT進展によってコード規模は増加し続けており，これまでの三菱電機の解析実績から，コード規模が５倍になると，誤検出を含む検出警告数は約22倍となる傾向を得ている(**図２**)。検出警告数が増加すると全検出警告の確認は困難となる。また，多数の誤検出の警告が含まれている場合が多いため，重要な警告が埋没してしまう危険性が高くなる。

このため，問題箇所を的確に検出するには，多数の誤検

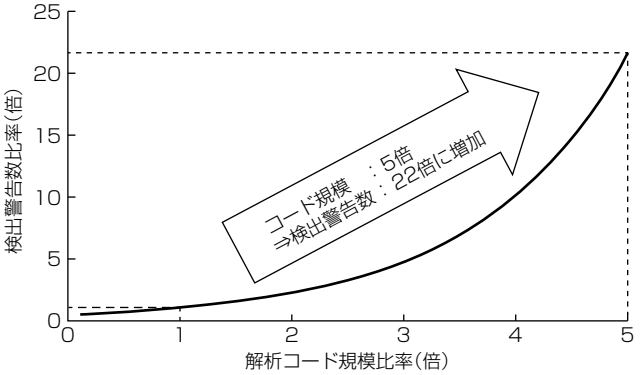


図２．コード規模と検出警告数の関係

表１．危険度と警告の対応付け（例）

危険度	影響度	発生確率	分類基準	静的解析出力警告
A	高	高～中	不具合の可能性が高く，最優先で確認・対応すべき警告	・ゼロ除算が行われる ・式の評価順序が未定義になる ・配列の領域外アクセスが発生する
B	高～中	高～中	不具合の可能性があり，確認・対応すべき警告	・異なるポインタ型間のキャストを実行している ・到達不能な文がある ・無限ループがある
C	中	高～中	確認・対応を推奨する警告	・shortからcharへの暗黙変換が行われる ・戻り値が無視される関数がある ・値を返すvoid関数がある
D	中～低	高～低	できれば確認することを薦める警告	・for文内でカンマ演算子が使われている ・long型からdouble型への型変換が行われている ・関数マクロが使われている
E	低	高～低	無視しても問題にはならないと思われる警告	・long long型が使われている ・main関数の戻り型が誤っている ・inlineが関数以外で指定されている

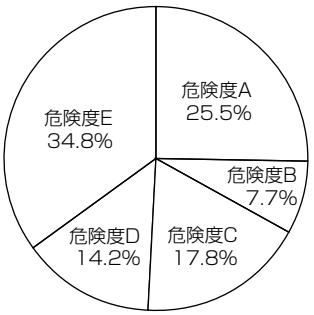


図３．危険度によって分類した警告の比率

出を含む大量の警告の中から問題顕在化の確率が高いものを効率的に絞り込むことが要求される。これに対して，これまでの静的解析の知見に基づき，影響度と発生確率の２つの観点から，危険度という５段階の尺度を新設し，警告をグループ化した（**表１**）。

⑴　影響度

検出された警告が実際に発生した場合にシステムが受ける悪影響を示す度合い（メモリ破壊，メモリリーク等の警告はプログラム実行が停止する等，致命的であり，影響度は高い）

⑵　発生確率

警告が指摘する内容が実際に発生している確からしさを示す度合い（main関数の戻り型の誤りを指摘する警告の発生確率は100％だが，配列の領域外アクセスの発生を指摘する警告の発生確率は50％に満たない）

この分類の適用によって，不具合の可能性が高く，最優先で確認，対応すべき危険度Aに含まれる警告（以下“危険度A警告”という。）は，検出される全警告の約25％に限定される（**図３**）。また，実際に解析を行った結果，危険度A警告だけを確認対象とした場合，検出された全警告を確認対象とした場合に比べ，確認すべき警告数は1/5となり，実際に解析を実施して検出された警告が不具合であった該当率（以下“ヒット率”という。）は６％から30％へ向上し，約５倍の検出効率改善を確認した（**図４**）。

3.2　ソフトウェア品質特性に対する充足度評価

現状のソフトウェア開発で，OSSや既存製品コードを流用するケースは95％を占める[3]。このため，流用開発の効率維持では保守性等のソフトウェア品質特性の確保が重要となってきている。

ソフトウェア品質特性はJIS X 25010：2013[4]で８種の品質特性が定義されており，コードの品質については信頼性，保守性，移植性及びセキュリティの４種の特性が強く関連する。このうち，セキュリティについてはネットワークに接続されるソフトウェアの増加に伴い，これらのソフトウェアを狙った攻撃によって機密データの消失や漏えい

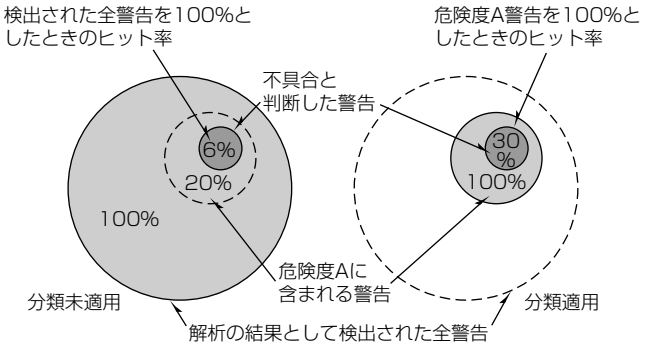


図４．危険度分類を用いた検出可能な不具合の抽出率

といった被害が発生している。このため，脆弱性等の問題があるコード記述を早期に発見して修正することが重要になってきている。そこで，これら４種の品質特性について次に示す取組みを実施した。

⑴　信頼性，保守性，移植性の取り組み

品質特性と，静的解析結果の警告及びメトリクスとを対応付け，二面からの充足度評価を可能とした（**表２**）。

⑵　セキュリティの取り組み

デファクトスタンダードとして活用が拡大しつつあるCERT-C／C＋＋コーディングスタンダード[5][6][7]（以下“CERT-C／C＋＋”という。）に記載されたセキュリティ上の欠陥を作り込ませないためのコーディングルールと静的解析の警告とを対応付けし，コードのセキュリティ品質特性の充足度の評価に加え，セキュリティに関連する問題の検出にも利用可能とした（**表２**，**表３**）。

この対応付けの適用によって，静的解析が出力する警告及びメトリクスから，流用開発の効率維持等で必要となるソフトウェア品質特性の評価，さらに，脆弱性等の問題があるコード記述の発見の効率性を向上させた。

4. む　す　び

大規模化するコードの誤検出警告による問題箇所の検出効率の悪化や，保守性やセキュリティ等のソフトウェア品質に関連付けたコードの品質評価の要求に対して，静的解析結果を選別して必要な情報だけを選別するスクリーニン

表2．ソフトウェア品質特性と静的解析のメトリクス及び警告の対応付け(例)

品質特性	品質副特性	具体的な特性	静的解析出力警告	静的解析出力メトリクス	
				名称	推奨値
信頼性	障害許容性 (耐故障性) 回復性	・エラー処理がきちんと実装されているか	・if-else-if文にelse節が又はswitch文にdefaultラベルがない ・例外処理が適切に行われてない		
保守性	解析性 モジュール性 再利用性 修正性 試験性	・ソースコードが読みやすいか ・全てのソースコードの書き方が統一されているか ・関数等の独立性が高いか ・試験が容易にできるか	・制御文の本体は中括弧で括る ・switch文のcaseがbreak文で終わっていない ・変数のスコープが適切でない ・１つのソースファイル内だけで使用される関数がstaticでない	複雑度	10以下
				マイヤーズインターバル	10以下
				関数行数	50以下
				推定静的パス数	200以下
				使用goto文の数	0
				凝集度	大きいほど良い
				結合度	小さいほど良い
				ネスト段数	
				外部変数の数	
				クラス継承段数	
				子クラスの数	
				親クラスの数	１以下
移植性	適応性	・コンパイラやCPUが変わった場合，修正箇所が少ないか	・言語規約に記載されている未定義，未規定，処理系定義の機能を使用している ・エンディアン依存の処理がある		
セキュリティ	機密性 インテグリティ 否認防止性 責任追跡性 真正性	・想定外の条件を含め，全ての条件に対応する処理が記述されているか ・CERT-C／C++が遵守されているか	・if-else-if文にelse節が，又はswitch文にdefaultラベルがない	複雑度	10以下
				推定静的パス数	200以下

表3．CERT-C／C＋＋コーディングルールと静的解析警告の対応付け(例)

CERT-C／C++コーディングルール			静的解析出力警告
種別	ID	概要	
プリプロセッサ	PRExxx-C	字句の結合や文字列化を行う際のマクロ置換動作をよく理解する	・"##"演算子の使い方が誤っている
宣言と初期化	DCLxxx-C	適切な記憶域期間を持つオブジェクトを宣言する	・関数内の自動変数のアドレスが外部ポインタ変数や戻り値等で関数外に持ち出されている
	DCLyyy-CPP	領域確保／解放を行う対となる関数は同じスコープでオーバーロードする	・対となるnew演算子又はdelete演算子がない
整数	INTzzz-C(CPP)	整数変換によってデータの消失や解釈間違いが発生しないことを保証する	・指定型サイズで表現不能な値が変数に設定されている
配列	ARRzzz-C(CPP)	配列以外のオブジェクトを指すポインタに対して整数の加算や減算を行わない	・正しくないポインタ計算が実施される ・ポインタと整数の加減算が実施される
オブジェクト指向プログラミング	OOPyyy-CPP	自分自身への代入を適切に扱う	・コピー演算子でコピー元と先のポインタが検査されていない
雑則	MSCyyy-CPP	疑似乱数の生成にstd::rand()関数を使用しない	・rand()関数が使用されている

グ環境を整備するとともに，プログラム実行停止等の重大な問題箇所を絞り込む方法，静的解析結果の警告とメトリクスを組み合わせてソフトウェア品質特性への充足度を評価する方法について述べた。

今後，IoT機器のコード規模の増加，流用開発の進展，ソフトウェア品質特性についても新たなセキュリティへの対策等が更に求められていく。

これらに対し，的確性に曖昧さが残る論理演算に起因する問題箇所の検出や，CERT-C／C＋＋以外のセキュリティについての規格化への追従を図り，本稿で述べた環境，概念をさらに進化させ，ソフトウェア実装品質の効率的な確保，向上に努めていく。

参 考 文 献

(1) (独)情報処理推進機構：ソフトウェア開発データ白書2007～2009／2010～2011／2012～2013／2014～2015
(2) 中岡邦夫：製品の設計初期段階で品質を作りこむ設計検証技術，三菱電機技報，87，No. 4，2～7(2013)
(3) (独)情報処理推進機構：組込みソフトウェア開発データ白書2015 (2015)
(4) JIS X 25010： 2013(ISO/IEC 25010： 2011)：システム及びソフトウェア製品の品質要求及び評価(SQuaRE)－システム及びソフトウェア品質モデル
(5) JPCERTコーディネーションセンター：CERT Cコーディングスタンダード
https://www.jpcert.or.jp/sc-rules
(6) Carnegie Mellon University/Software Engineering Institute：SEI CERT C Coding Standard
https://www.securecoding.cert.org/confluence/display/c/SEI+CERT+C+Coding+Standard
(7) Carnegie Mellon University/Software Engineering Institute：SEI CERT C++ Coding Standard
https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637