ソースコード静的解析技術

Static Source Code Analysis Technology

古賀　国秀　　山元　和子
■ KOGA Kunihide　　■ YAMAMOTO Kazuko

ソフトウェア規模の増加に伴い，様々なスキル（能力）の技術者がプログラミングを行うようになった。その結果，コーディングスキルを底上げする必要性が高まっている。

東芝は，標準コーディング規約を策定するとともに，コーディングルールチェック静的解析ツールを16部門に導入した。更に複雑な設定作業を自動化し，簡単に利用できるフロントエンドツール(注1)を開発しこの静的解析ツール導入を加速させた。また，従来は動的解析でしか解析できなかったメモリ関連のバグを検出する新しいタイプの静的解析ツールの導入も，5部門で推進している。

The amount of source code in software is increasing so rapidly that developers with different programming skill levels have to write the source codes. As a result, there is an increasing need to enhance the level of coding skills.

In order to normalize such coding skills, Toshiba has formulated a coding standard for C language, and has also introduced a static analysis tool into 16 departments of Toshiba. In addition, we have developed a front-end tool for static analysis to simplify and automatically generate analysis configurations. Recently, we have been promoting the introduction of a new type of static analysis tool, which can detect memory-related bugs that could previously only be detected by dynamic analysis, into five departments.

1　まえがき

ソフトウェアの開発における品質向上，期間の短縮，及びコスト削減の要求は，年々厳しさを増している。この課題に対して，実装工程でソースコードの品質を高めることが，問題解決の一つの手段である。静的解析ツールは，その解決策を提示できる強力なツールである。静的解析ツールとは，実際にプログラムを動かすことなく静的にソースコードを解析して，品質上の問題点となる箇所を指摘するツールである。

東芝は，以下に示すソースコード品質の二つの問題に焦点を当てて対策を講じた。

第一の問題は，コーディングスキルに格差がある点である。大規模開発では，一定レベル以上のコーディングスキルを持つ者だけで開発することは困難であり，一部のコーディングスキルの低い者がソースコード全体の信頼性を下げてしまう。この問題に対しては，当社標準コーディング規約を策定するとともに，コーディングルールチェック静的解析ツールの導入推進と高機能化を行ってきた。

第二の問題は，ソースコード中に含まれる条件分岐のパターンをすべて実行させて，ランタイムエラー（プログラム実行時に特定の条件下で発生するエラー）を引き起こすバグ（コーディングの誤りや欠陥）を検出することは，大きな工数が掛かるという点である。この問題に対しては，ランタイムエラーを検出可能な新しいタイプの静的解析ツールの導入を推進してきた。

ここでは，コーディングルールチェック静的解析ツールとランタイムエラーを検出可能な静的解析ツールの特長，及び導入成果について述べる。

(注1)　解析実行の自動化やWeb上での結果閲覧のほか，統計情報表示，ユーザー認証，各種設定作業を行うツール。

2　コーディングルールチェック静的解析ツール

コーディングルールチェック静的解析ツールは，コンパイル時には構文エラーや警告にならない各種コーディング規約違反を検出できるのはもちろんのこと，バグに陥りやすい書き方を指摘することができる。開発担当者が，指摘された箇所を検討し，必要な修正を行うことによって，ソフトウェアの信頼性，保守性，移植性，及び効率性といった品質を確保することができる。

2.1　ツールの特長

コーディングルールチェック解析の方式は，パターンマッチングと呼ばれ，あらかじめ定義したコーディングルールに対してコーディングルール違反を起こしている箇所をチェックする方法である。例えば，**図1**のわずか10行余りのプログラムには，誤りが1か所，望ましくない書き方が2か所ある。

(1)　誤り箇所（1行目）　　マクロの引数をかっこで囲うべきである。引数をかっこで囲うことによって，マクロ引数

```
1行目:#define
        CHECK_COLUMN_SIZE(column_number)
        column_number & 07
2行目:// TAB文字をSpace8文字に変換する
3行目:void tabtospace( char c )
4行目:{
5行目:  int column = 0;
6行目:  if (c=='¥t') {
7行目:     do {
8行目:        putchar (' ');
9行目:        column++;
10行目:     } while
        (CHECK_COLUMN_SIZE(column) != 0);
11行目:  }
12行目:}
```

図1. 悪いプログラムの例 ― プログラム中のルール違反箇所を解説することで，静的解析の機能を知る。

Example of bad code

自身が式である場合に起こりうる問題を回避できる。

ただし，今回は引数は整数であり，この問題は発生しない。しかし，プログラムを変更し引数に式を入れた時点でバグとなる。

修正例：1行目:#define
CHECK_COLUMN_SIZE(column_number)
(column_number) & 07

(2) 望ましくない書き方（10行目）　演算子の優先度はISO（国際標準化機構）標準規格で完全に定義されているが，それを覚えるのは簡単ではない。かっこは，演算子が評価される順序を明示的にするうえで必要となる。10行目のマクロを展開すると以下のようになる。

10行目:| while(column & 7 != 0)

ビット演算子'&'より比較演算子'!='が優先度が高いため"column & (7 != 0)"となり，7と0を先に比較してしまう。熟練者でもかっこを付ける習慣がこのリスクを遠ざけるのである。

修正例：1行目:#define
CHECK_COLUMN_SIZE(column_number)
((column_number) & 07)

(3) 望ましくない書き方（10行目）　10行目にはもう一つの問題がある。int（整数）型のcolumnをビット演算していることである。ISO標準規格で符号付きデータ型のビット単位の演算（|，&，^，～）は規定されていない。整数の負の値は通常 2 の補数で表現していることは知られているが，これは規格で定義されているわけではないことはあまり知られていない。多くのコンパイラは期待される処理を行うが，ビット単位の演算で符号付きデータを操作することは移植性を損なう。

修正例：5行目: unsigned int column = 0;

このようなバグに陥りやすい書き方を数百万行のソースコードから人手によるコードレビューで発見することは難しいが，静的解析ツールであれば，これらすべてを検出し，これまで気づかなかった単純ミスやコーディングミスを早期に発見し，テスト工程以降に持ち込まないようにできるという，とても大きなメリットがある。

2.2 フロントエンドツールの開発

コーディングルールチェック静的解析ツールの一つであるQAC(注2)の導入支援を，2006年度から東芝グループ向けに実施している。

QACは，トップクラスの静的解析の能力を持っているが，利用者が使いこなすためには複雑な設定手順の習得と設定作業に時間を要するツールである。例えば，デジタルコンシューマ製品の開発では，数百万行のソースコードを解析するたびに，ツールへの各種設定作業と解析作業を合わせて14人日を要する。

そこで当社は，QACの複雑な設定作業を自動化し簡単に利用できるフロントエンドツールを開発した。バックエンドの解析エンジンとしてはQACを利用しているが，利用者はQACの存在を意識することなく，Webブラウザだけで解析を行えるようになり，解析時間は約70％削減された。

このように解析実行の自動化，複雑な設定作業の簡素化，及びWeb上での閲覧を可能とすることで効率化を達成してきた（**図2**）。更に，ユーザー登録や，ソースコードのアップロード，セキュアな（安全性が保証された）実行などの機能を追加し，利用者だけでなく運用管理者にも容易に作業を実施できるよう改良した。また，QAC以外の自社製や他社製のツールをプラグインし解析する機能を備えている。

2.3 ツール導入の推進方法

使い勝手はフロントエンドツールで向上したが，導入を進めるためには，違反箇所を過剰に指摘したり，エラーではない箇所をエラーであると誤って指摘する誤検知という問題点を

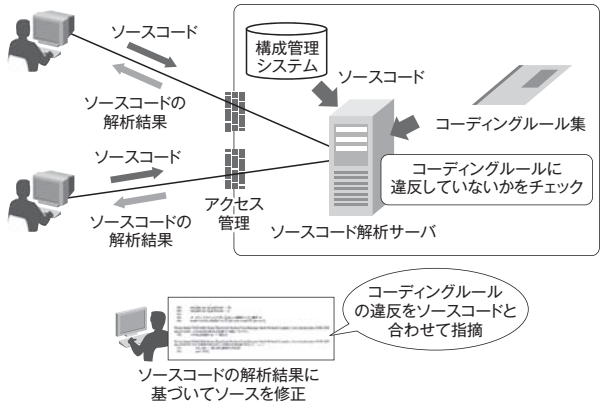

図2. コーディングルールチェック静的解析ツールの運用イメージ ― フロントエンドツールにより，Web上で，ソースコードのアップロードや解析結果の閲覧を行う。

Image of source code static analysis system operation

(注2) QACは，英国Programming Research社の商標。

解決する必要があった。そのため，導入サポート体制を整備した。

東芝グループでは，静的解析ツールの導入サポートを利用することにより，開発部門が複雑かつ高度な活用ノウハウを習得する必要なく，必要に応じてきめ細かいサポートを受けることができるようになった。導入サポートの特長は，次のとおりである。

(1) リスクレベルにより指摘内容を分類し，過剰指摘のない結果を出力する。
(2) 誤検知を切り分ける。
(3) あらゆるコンパイル環境に対応し，すべてのソースコードを解析可能にする。
(4) 指摘箇所を詳細に分析した品質評価レポートを作成する。
(5) 定期的な解析作業を代行する。

これらにより，導入・維持コストが下がるとともに，より効果的な活用が可能になり，製品の品質向上に寄与している。

3 ランタイムエラーを検出可能な静的解析ツール

最近，新しいタイプの静的解析ツールがリリースされ，注目されている。新しいタイプの静的解析ツールは，主に実行時に発生するメモリ関連のバグを中心に静的に検出する。検出可能なバグはツールによって異なるが，主に次のようなランタイムエラーを引き起こす重大なバグを検出できる。

(1) メモリの未解放（メモリリーク）
(2) NULL(注3)ポインタの参照
(3) 配列の領域外のアクセス

以下に，ツールの特長とバグ検出の方法について述べる。

3.1 ツールの特長

従来，ランタイムエラーは動的解析ツールによって検出していた。動的解析ツールを使用してバグを検出するには，ソースコードをビルド(注4)後実行する。動的解析ツールは，エラーが発生した所で実行を停止し，ソースコードの該当箇所を表示する。

動的解析ツールと比較した場合，静的解析ツールの主なメリットは三つある。

(1) 網羅性が高い　　動的解析ツールは，実行された箇所についてだけバグ検出が可能であるため，テストの網羅率が低いと検出されないバグが残ってしまう。一方，静的解析は後述する動作原理に従い，実行可能なパスは網羅的にチェックし，バグを検出することが可能である。しかも，テストケースを準備する必要がない。
(2) 適用可能な実行環境が多い　　動的解析ツールは，サポートする実行環境がほぼWindows®(注5)やLinux(注6)に限られてしまう。組込みソフトウェアはこれら二つ以外の実行環境で実行される場合も多く，従来，動的解析ツールを適用できないケースが多かった。一方，静的解析ツールはコンパイル時にバグ検出を行うので，実行環境には依存せず幅広いソフトウェアに適用可能である。
(3) 大規模データへのスケーラビリティ（適用可能性）が高い　　動的解析ツールはエラーチェックをしながら実行するので実行時間が数倍～10倍以上にもなり，もともとの処理負荷が高いソフトウェアでは，実用的なテストが不可能なほど処理負荷が高くなってしまう問題がある。一方静的解析ツールは，もともとのビルド時間の数倍程度で解析でき，大規模なソフトウェアでも解析不能になることはない。

このように，静的解析ツールは優れた点がある反面，デメリットも二つある。

(1) 誤検知率が高い　　動的解析ツールでは誤検知率はゼロに近いが，静的解析ツールの誤検知率は，当社のベンチマーク（比較のための指標）で有効だと判断したツールであっても15～35％であった。誤検知の多さは，バグを修正する開発者の負担を増加させるため，低く抑えることが重要である。
(2) 価格が高い　　数百万～数千万円と高額である。現在発売元は5社程度に限られ，また，商用ツールに匹敵するような機能を持つフリーのツールは存在しない。

3.2 バグ検出の方法

ランタイムエラーを検出可能な静的解析ツールは，**図3**のような処理でバグ検出を行う[1]。

まずビルドのプロセスを監視し，ビルドされるソースコードやビルド条件などの情報を取得する。これらの情報を利用し，中間モデルを作成する。中間モデルは抽象構文木(注7)，コー

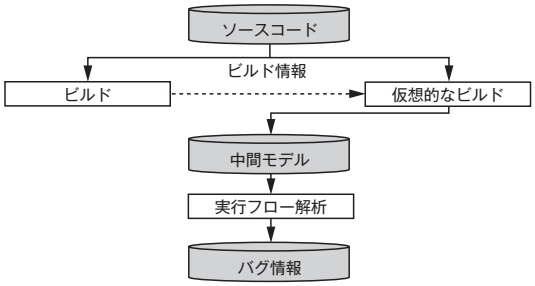

図3．静的解析ツールの概略処理の流れ ― ビルド時に実行フロー解析を行うことで，ランタイムエラーの原因となるバグを検出する。
Flow of static analysis

(注3) ヌルと呼ぶ。プログラミング言語やデータベースのデータ表現の一つで，何のデータも含まれない状態。
(注4) ソースコードのコンパイルやライブラリのリンクなどを行い，最終的な実行可能ファイルを作成すること。
(注5) Windowsは，米国Microsoft Corporationの米国及びその他の国における商標又は登録商標。
(注6) Linuxは，Linus Torvalds氏の米国及びその他の国における登録商標。
(注7) コンパイラが生成するプログラムの内部表現。

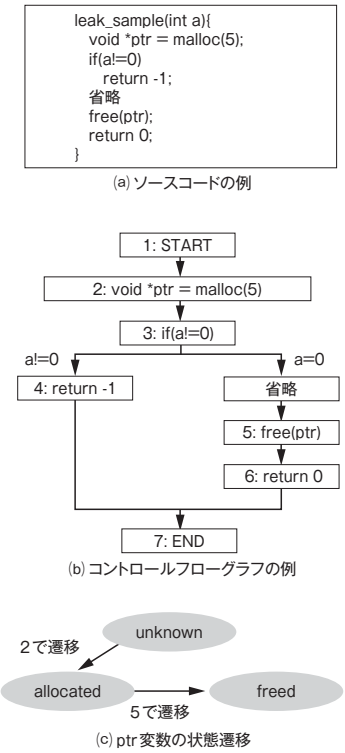

図4. フロー解析によるメモリリークの検出例 — ソースコードをコントロールフローグラフによる中間モデルに変換し，中間モデル上で注目する変数の状態遷移をチェックすることで，バグを検出する。

Example of memory leak detection by flow analysis: (a) example of source code, (b) example of control flow graph, (c) state transition of variable ptr

ルグラフ(注8)，制御フローグラフなどである。中間モデル上で実行可能なパスを網羅的にチェックすることによって，バグ検出を行う。検出されたバグは，最後にWebブラウザを介してユーザーが閲覧可能な形で出力する。

実行可能なパスをたどってバグを検出する処理につき，**図4**を用いて述べる。

このソースコードには二つの実行可能なパスがあり，そのパスに沿って注目する変数ptrの状態をチェックする。1→2→3→5→6→7のパスでは，freed（メモリが解放された状態）でENDに達するのでメモリリークは発生しないが，1→2→3→4→7のパスでは，allocated（メモリが確保された状態）でENDに達するのでメモリリークが発生する。この例ではパスが関数内で閉じているが，関数内から関数をコールする構造になっている場合は，関数をまたいで同様の解析を行う（プロシージャ間解析）。

このように，どのような実行パスの中でバグが発生するのかを提示するため，開発者はバグの原因を特定し，修正することができる。

（注8） プログラムの関数の呼出し関係を表すグラフ。
（注9） Preventは，米国Coverity社の商標。

4 導入成果

コーディングルールチェック静的解析ツールは，フロントエンドツール及び導入サポートによる簡便化と導入・維持コストの低減により，2007年度以降，飛躍的に導入を拡大することができ，東芝グループの16部門で活用されている。

また，ランタイムエラーを検出可能な静的解析ツールは，当社ベンチマークによりトップクラスの能力を示したPrevent™(注9)の導入が2008年8月から開始され，既に5部門で活用されている。導入直後から致命的なバグの検出に貢献し，テストでは発見困難であったハングアップを究明するという実績も上げている。製品サイクルの短いデジタルコンシューマ製品では，1千万行を超えるソースコードを週1回解析する運用体制を確立した。また，これらの部門では，コーディングルールチェック静的解析ツールと併用することで，より大きな効果を上げている。

5 あとがき

コーディングルール チェックとランタイムエラーの検出という二つの静的解析ツールの特長について述べるとともに，当社の複数の部門で活用され，ソースコードの品質向上に貢献している状況について述べた。

静的解析ツールは，ソースコードの品質向上に非常に有用ではあるが，まだまだ発展途上であり，検出可能な領域は日進月歩でより広く，より深くなってきている。

当社は，最先端の静的解析技術を常に追い求め，自社製，他社製を問わず複数のツールを活用し，漏れのない包括的な静的解析技術を東芝グループ全体に普及展開していく。

文 献

(1) Engler, D., et al. "Checking system rules using system-specific, programmer-written compiler extensions". 4th Symposium on Operating System Design & Implementation. San Diego, California, USA, 2000-10, USENIX. 2000.

古賀 国秀 KOGA Kunihide
ソフトウェア技術センター プロセス・品質技術開発担当主務。ソフトウェアの品質技術開発に従事。
Process and Quality Technology Group

山元 和子 YAMAMOTO Kazuko
ソフトウェア技術センター ソフトウェア設計技術開発担当主務。ソフトウェア開発インフラの技術開発に従事。
Software Design Technology Group