卒 業 論 文 概 要 書

CD

2009 年　2 月提出　　学籍番号 1G05R080-5

学科名	コンピュータ・ネットワーク工学科	氏 名	坂本　一憲	指 導 教 員	鷲崎　弘宜　准教授
研 究 題 目	多言語対応のテストカバレッジ測定フレームワークに関する研究				

1. はじめに

テストカバレッジ(網羅率)とは，テスト可能なプログラムのソースコードが，テストされた割合を示す．

テストカバレッジには，ステートメントを対象とする命令網羅(C0)，条件分岐を対象とする分岐網羅(C1)など様々な種類があり，ソフトウェアテストの目的と用途に応じて，適切なものを選択する．

近年，プログラミング言語の多様化，多言語のソフトウェア開発の普及に伴い，既存のテストカバレッジ測定ツールでは様々な問題が生じている．

本研究では，多言語に対応したテストカバレッジ測定フレームワークを提案して，容易に対応言語を追加することができる，統一的で柔軟なテストカバレッジ測定ツールの開発を実現する．

2. 既存のテストカバレッジ測定ツールの問題点

2.1. 莫大な開発コスト

測定ツールは，構文解析器，意味解析器などの複雑な機能要素から構成される．また，既存の測定ツールは主要な言語のみに対応する．

そのため，COBOL言語など，レガシーな言語に対応する測定ツールの開発や，Java言語の1.4から5.0へのバージョンアップなど，言語仕様の変化に伴う修正は必要となるが，莫大なコストがかかる．

2.2. 統一性を欠いた測定

既存ツールは，各言語に特化したものが多く，複数言語によるソフトウェア開発では利用しにくい．

例えば，Java言語とPython言語を用いてサーバークライアントモデルのソフトウェアを開発する場合，Javaに対応するCobertura[1]とPythonに対応するStatement Coverage for Python[2]を組み合わせる必要がある．しかし，後者は命令網羅のみで，結合テストにおいて分岐網羅を測定できない．

2.3. 柔軟性を欠いた測定

既存ツールは，カバレッジの測定対象や範囲を柔軟に指定できず，柔軟な測定ができない．

例えば，特定のライブラリ呼び出しのみをテストする場合でも，既存ツールでは全ステートメントを測定対象としてしまう．そのため，テストの網羅性を正確に判断できない．

2.4. 不完全な測定

既存ツールは，測定対象に漏れがある場合がある．

例えば，Coberturaでは，実行可能なバイナリを参照して測定するため，コンパイラによって削除されたデッドコードを測定対象に含むことができない．

3. テストカバレッジ測定フレームワークの概要

本研究では，複数のプログラミング言語対応のテストカバレッジ測定フレームワークを提案する．

フレームワークでは，まず，抽象構文木を介して，ソースコードにテストカバレッジ測定用コードを埋め込む．次に，埋め込み済みソースコードを任意の処理系で実行する．最後に，実行した際に得られた情報をもとに，測定結果をユーザーに表示する．

なお，埋め込むコードは，副作用がなく，ソースコードの意味を変化させない．このコードは，測定対象が実行さるたびに，その情報をファイルに出力する．それによって，テストカバレッジを測定する．

3.1. フレームワークの構成

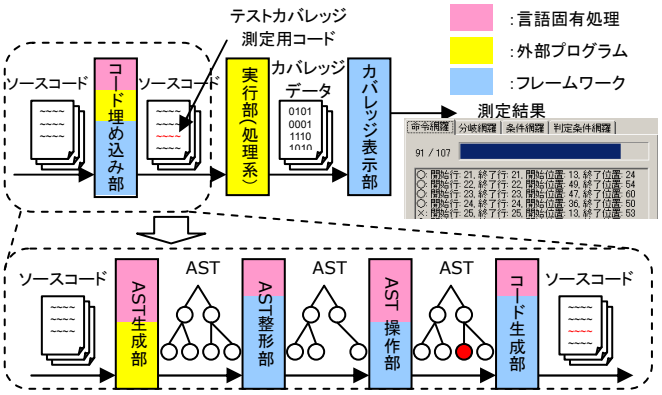

図1. フレームワークの構成.

3.2. コード埋め込み部（四つの構成要素）

AST生成部：ソースコードを入力してASTを生成する．この処理は，各言語に強く依存するため，言語に特化した実装が必要である．しかし，構文解析器を自動生成するコンパイラコンパイラや，ライブラリを利用することで，開発コストを低減できる．

AST整形部：冗長要素の削除や，補助要素の追加などを行う．例えば，得られたASTの抽象度が低い場合，冗長な要素を削除することで，AST操作部の実装を簡略化する．言語特有の処理が少ないため，フレームワークが大部分の実装を提供する．

AST操作部：要素の列挙，挿入，削除，置換により測定用コードを埋め込む．例えば，図2のように，命令網羅の測定で，各ステートメントの要素を列挙して，その直前に測定用コードの要素を挿入する．

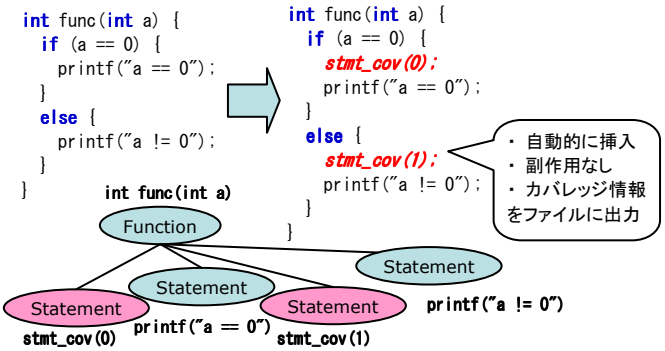

図2. 測定用コードの埋め込み例

さらに，各操作を組み合わせて，より複雑な操作を実現する．フレームワークは，共通処理の実装と処理の組合せ支援により，開発コストを低減する．

コード生成部：ASTを入力してソースコードを生成する．ASTでは，元のソースコードのトークンを記憶していることが多い．そのため，ほとんどの処理は共通化できる．フレームワークは，特定の非終端記号に対する処理の実装のみを要求する．

3.3. コード実行部

埋め込み済みコードを実行して，測定に必要な情報を収集する．ユーザーは任意の処理系を用いて，得られたソースコードでテストを行えば良い．

3.4. 測定結果表示部

収集した情報から測定結果の表示を行う．フレームワークは，測定結果表示部の完全な実装を提供する．開発者は独自に改良も可能である．

4. テストカバレッジ測定フレームワークの実装

フレームワークは.NETで開発した．Dynamic Language Runtimeも搭載しており，.NETで実装したアセンブリか，対応スクリプト言語で，言語固有の処理の追加や機能拡張が可能である．

本研究では，フレームワークを用いて，Java，Python，Cの3言語に，命令網羅，分岐網羅，条件網羅，分岐／条件網羅の4種類のカバレッジで対応するツールを実装した．なお，図3はPythonの命令網羅を実装した例とクラス図である．例では，埋め込み位置を列挙する処理を簡潔に実装できている．

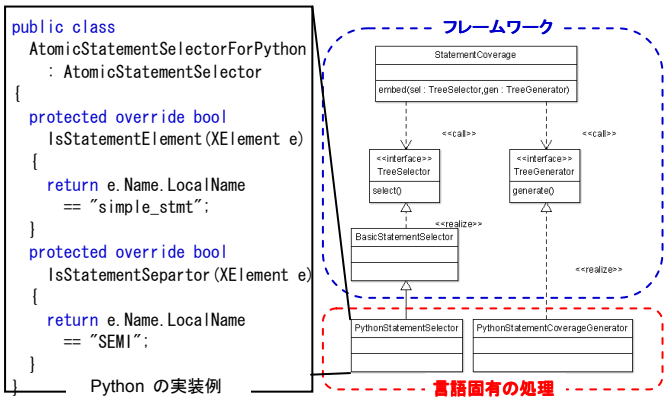

図3. Python対応の実装例

5. 評価

フレームワークを利用した実装例と，Cobertura，Statement Coverage for Pythonで比較を行う．

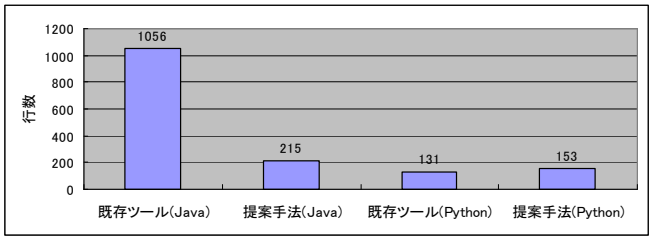

図4. 埋め込み処理の言語固有な実装のコード行数

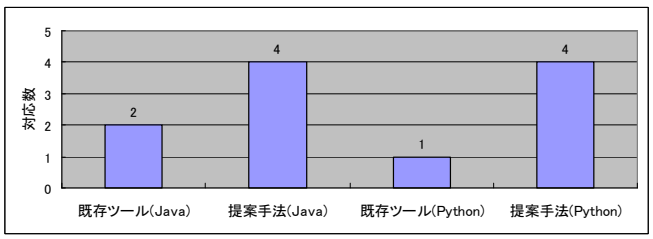

図5. 対応するテストカバレッジの種類数

図4のコード行数では，Javaへの対応において，実装例の方が80%も行数が少ない．一方，Pythonへの対応では，実装例の方が若干劣っている．

しかし，図5の種類数では，実装例が最も多く，Pythonに対応する既存ツールが最も少ない．そのため，種類数を考慮すれば，フレームワークを利用した実装例が最も良い結果となっている．

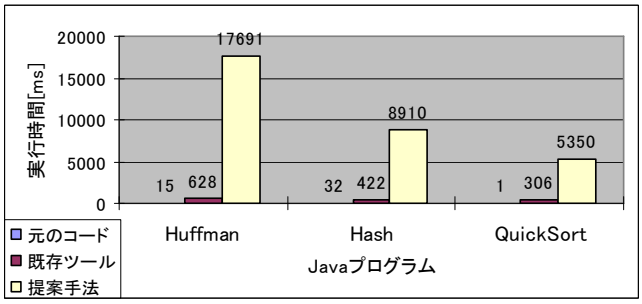

図6. テスト時のパフォーマンス

図6では，Coberturaと比較して実装例の方が，10倍から30倍程度性能が低下している．これは，埋め込むコードが，実行するたびにファイル出力を行うという，パフォーマンスを考慮しない実装に起因する．そのため，共有メモリへの書き込みなど高速な出力処理に置き換えることで改善できる．

最後に，3種類のプログラミング言語への対応，それぞれ4種類のテストカバレッジへの対応，測定対象を変更する機能拡張の実現を踏まえて，統一的で柔軟な測定を実現した．また，ソースコードに測定用コードを埋め込むことで，デッドコードも測定範囲に含めた完全な測定を実現した．

6. 関連研究

桐井ら[3]は，ソースコードに測定用コードを埋め込むことで，測定ツールを実装する手法を提案している．この手法では，4種類の言語に対応した例が挙げられているが，本研究とは異なり，新たに対応言語を増やす際の支援や，柔軟な測定はできない．

Hridesh Rajanら[4]は，アスペクト指向プログラミング言語を応用して，測定対象を柔軟に指定する手法を提案している．この手法では，1種類の言語に特化した例が挙げられているが，本研究とは異なり，複数言語に対応する統一的な測定はできない．

7. おわりに

本研究では，多言語対応のテストカバレッジ測定フレームワークを提案した．複数のプログラミング言語への統一的な対応，共通処理の再利用による開発コストの低減，機能拡張の支援による柔軟な測定，ソースコードへの埋め込みによる完全な測定を全て実現する．今後の課題としては，パフォーマンス改善とプロファイリングへの応用が挙げられる．

参考文献

[1] Cobertura, http://cobertura.sourceforge.net/
[2] Statement Coverage for Python, http://garethrees.org/2001/12/04/python-coverage/
[3] 桐井隆志，三好辰弥，岸上諭，大里立夫，曽根原勝，"ソースコード埋め込み型カバレッジツールについて"，情報処理学会第 65 回全国大会, 2003.
[4] Hridesh Rajan and Kevin Sullivan, "Aspect Language Features for Concern Coverage Profiling", International Conference on Aspect-Oriented Software Development, 2005.