プログラム実行時の状態を再現する単体テスト支援 —レガシーコードに対するテストケース自動生成—

Unit Testing Support by Recreating the Status of Program Runtime: Automatic Test Case Generation in relation to Legacy Codes

大谷雄平† 橋浦弘明‡ 古宮誠一‡
Yuhei Otani† Hiroaki Hashiura‡ Seiichi Komiya‡

†芝浦工業大学　工学部
‡芝浦工業大学大学院　工学研究科
†Faculty of Engineering, Shibaura Institute of Technology.
‡Graduate School of Engineering, Shibaura Institute of Technology.

要旨

ソフトウェア開発においてソースコードに単体テストを書くことは重要であるが, この作業は容易ではない. 特に, ファイルやデータベースといった外部のリソースを扱うプログラムの単体テストを作成するには，読み込むリソースの状態を設定する際に多くのセットアップを必要とするため，より一層困難になる．本稿では，プログラムの実行履歴からオブジェクトの生成方法やオブジェクトの変化の流れを追跡することにより，プログラム実行時の状態を再現する単体テストの自動生成手法を提案する．これにより，外部リソースを扱うプログラムなど，単体テストを作成することが困難なプログラムに対し，自動的に単体テストを生成することが可能になる.

1. はじめに

ソフトウェア開発において，ソースコードにテストを書くことは重要な事柄の 1 つであるが，テストのないコードが世の中には存在する．テストのないコードのことをフェザーズは”レガシーコード”[1]と定義し，レガシーコードを扱うためには，開発中すぐにフィードバックを得ることができる単体テストを書くことが重要であると述べた．しかし，レガシーコードに対し単体テストを書くことは困難であるという問題がある．具体的には，テストを書かないコードは内部構造が複雑に絡み合っていて，読みにくいコードになっている場合が多いからである.

そこで本稿では，java プログラム実行時の履歴を基にした単体テストの自動生成手法を提案し，単体テストプログラムを自動生成するシステムを開発する.

2. 単体テストを書くことが困難な理由

レガシーコードに単体テストを書くことが困難である理由として，次の 3 点を挙げる.

1 つ目は，外部リソース（ファイル・データベース・ネットワーク等）を用いるプログラムをテストする場合，リソースの状態を正しく設定することが難しいという問題である．例えば，図 1 のようにファイルのストリームオブジェクトを引数に受け取り, ファイルを読み込んで処理を行うメソッドがある. このメソッドはファイルの 4 行目の値 d から読み込んで処理を行うことを期待している（前提条件）．しかしながら，ファイルとのストリームを生成した時点では，ファイルの先頭からしか読み込むことができない（図 2)．このため，前提条件を満たすためには，ストリームの内部状態をファイルの 4 行目から読み込むことができるように設定する必要がある．設定方法の 1 つとして，実行途中のストリームの内部状態を切り取り，それを再現する方法が考えられる．通常，実行途中のオブジェクトの状態を再現するにはシリアライズを用いる．しかし，入出力ストリーム関連などのシリアライズ可能でないオブジェクトはこの方法を使用できないため，ストリームの状態の再現は困難な作業である.

2 つ目は，テスト対象のクラスが多数の依存関係を持っている場合，そのクラスを動かすことが難しいという問題である．コンストラクタの引数に多くのオブジェクトを必要としたり，生成方法の難しいオブジェクトを必要としたりすると，クラスを動かすために数多くのセットアップが必要になる.

3 つ目は，メソッドに副作用がある場合，その副作用の影響範囲を調べることが難しいという問題である．メソッドの事前条件や事後条件として，メソッドがオブジェクトの値をどのように変化させたかを調べる必要があるが，レガシーコードは内部構造が複雑なため，メソッドがオブジェクトに与える副作用の影響範囲を手作業で調べていくことは困難な作業である．

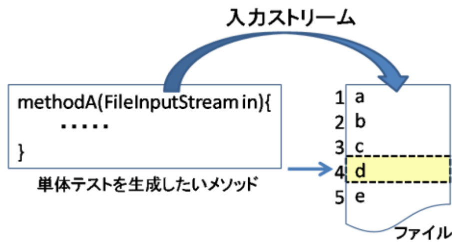

図 1 前提条件を満たすストリームの状態

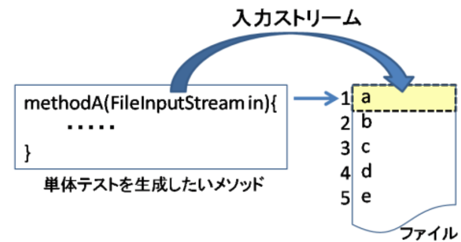

図 2 ストリームの初期状態

3. 関連研究

Elbaum ら[2]は XStream というライブラリを使ってメソッド前後のオブジェクトの状態を保存し，それらを復元することでシステムテストケースから部分的な小さなシステムテストを自動生成することを可能にした．しかし，XStream はシリアライズ可能でないオブジェクトは扱えないため，入出力関連のオブジェクトを扱うプログラムは対象外である．本研究では入出力関連のオブジェクトも再現可能な単体テストの自動生成手法を提案する．

4. 提案する手法

前述の問題を解決するために，プログラムにテストデータを流して実行履歴を取得し，それを組み立てて単体テストを自動生成する手法を提案する．実行履歴を用いることで，外部リソースとのストリームオブジェクトに与えた操作が全て明らかになるため，与えた操作を全て実行することにより，ストリームの状態を自動的に再現することが可能になる．また，実行履歴からオブジェクトの生成方法が判明するため，実行履歴の通りにオブジェクトを生成していくことで，依存関係を解くことができる．さらに，メソッド内でオブジェクトに与えた副作用も実行履歴を見れば明らかである．この手法は次の手順で実現する．

手順1.　要求仕様書から，プログラム全体を動かすためのテストケースを人手で作成する
手順2.　手順 1 で作成したテストケースをプログラムに流して実行履歴を取得し，図 3 で示すような情報を得る
手順3.　実行履歴を解析し，JUnit[3]の単体テストプログラムを自動生成する

この手法の前提として，プログラムにはバグがなく実行して動かせる状態であるということと，手順 1 で作成するテストケースのために，要求仕様書が存在していてプログラムの動かし方は分かるという条件を想定する．

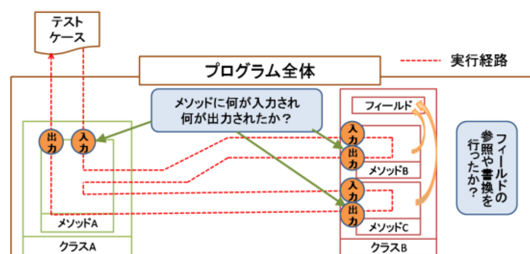

図 3 実行履歴から得られる情報の一部

4.1. 実行履歴の取得

プログラムの実行履歴は traceglasses[4]を用いて取得する．traceglasses は，java プログラムの実行をトレースとして記録し，利用者が対話的な欠陥の追跡をすることのできるデバッガである．実行されたプログラムの記録を，最初から最後まで可能な限りトレースとして記録しているため，オブジェクトの生成方法や，オブジェクトが行った操作などを全て追跡することができる．本手法に必要な実行履歴は，この traceglasses で取得したトレースを用いる．

4.2. 単体テストプログラム

JUnit の単体テストは，メソッド単位で行われる．単体テストプログラムの流れは次のようになる．

まず始めに，テスト対象のメソッドを実行するためのセットアップを行う．具体的には，メソッドを呼び出すために必要なオブジェクトを生成し，テスト条件を満たすようにオブジェクトの値をセットする．次に，メソッドの事前条件を検査する．事前条件の検査は，メソッド実行後にオブジェクトの値が予測通り変化したかどうか，変わる必要のないオブジェクトの値が変化していないかどうかなどを判定するために行う．次に，メソッドを実行し，メソッドの事後条件を検査する．本手法は，これらを実現するコードを全て実行履歴から生成する．

4.3. 実行履歴の解析

実行履歴は図 4 のような流れで解析を行う．

まず始めに，実行履歴を最初から最後まで調べ，標準ライブラリを除く通過したメソッドの一覧を取得する(Ⅰ)．このフェーズでは各メソッドについて次の事項を調べる．

- メソッドを呼び出したオブジェクトの ID 番号(traceglasses では，使用したオブジェクトに ID 番号を付けて記録している)
- メソッド内で参照したオブジェクトの ID 番号と，参照したフィールドの名前
- メソッド内で変化したオブジェクトの ID 番号と，変化前と変化後の値
- メソッドに渡した引数と，メソッドが返した返り値

次に，Ⅰで得たメソッドを呼び出したオブジェクトの ID 番号と対応するオブジェクトが生成されている箇所を探索し，オブジェクトを生成する(Ⅱ)．オブジェクトを生成する際，引数にオブジェクトを渡している場合，そのオブジェクトの ID 番号を調べ，その ID 番号に対応するオブジェクトが生成されている箇所を探索し，新たにオブジェクトを生成する．このように，新たにオブジェクトが必要になる場合には，再帰的にオブジェクトが生成されている箇所を探索し，必要なオブジェクトを生成する(Ⅲ).

次に，生成した各々のオブジェクトについて，オブジェクトが生成された地点からメソッドにオブジェクトを渡した地点の間における，オブジェクトが行った操作を探索する．対象のオブジェクトから呼び出されたメソッドを全て実行することにより，オブジェクトをプログラム実行時の状態に再現することができる(Ⅳ)．Ⅰにおいてメソッド内でオブジェクトを参照しているか調べた結果，参照していないことが判明しているなら，オブジェクトの状態を再現しなくてもメソッドの挙動は変化しないため，Ⅳの作業は必要ない．Ⅱ～Ⅳの処理をすることによりセットアップ作業が完了する．

最後に事前条件や事後条件を検査するコードを生成する(Ⅴ)．具体的には，事前条件としてメソッド内で変化したオブジェクトの変化前の値を検査するコードを生成し，メソッドを実行するコードを生成し，事後条件としてメソッド内で変化したオブジェクトの変化後の値を検査するコードと，メソッドの引数と返り値を検査するコードを生成する．これらのコードの生成は，Ⅰによりメソッド内で変化したオブジェクトの値や，メソッドの引数と返り値を調べているため可能である．Ⅱ～Ⅴの処理により，一つのメソッドに対する単体テストプログラムの生成が完了する．これら一連の解析を，テストケースを流した際に通過したメソッドの単体テストプログラムが全て生成済みになるまで行う．

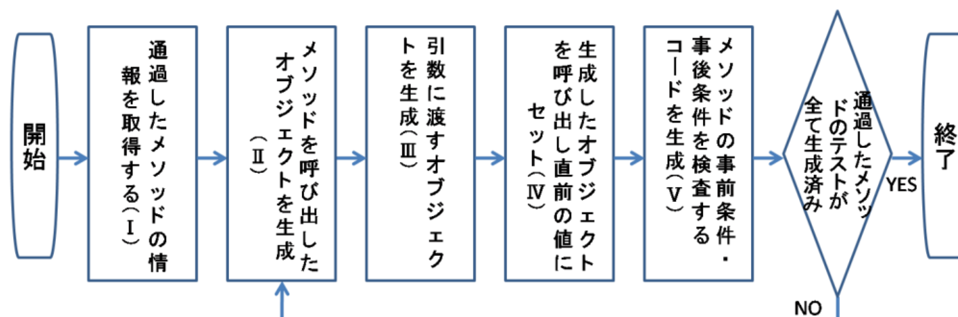

図 4 実行履歴解析の流れ

5. 評価について

本手法を適用して生成したテストケースに対し，次の評価項目について検討する．

- 全メソッド数に対する，テストしたメソッド数の割合
- テストしたメソッドの網羅率（命令網羅・分岐網羅）
- テストの成否の割合
- テストプログラムを生成する際にかかった時間
- テストプログラムを実行する際にかかった時間

実験として，500 行程度の小さなプログラムに対し，同一のシステムテストケースから本手法を適用して生成した単体テストケースと，人手で作成した単体テストケースを用意し，評価項目について比較を行う．最終的にはもう少し大きなプログラムに対し，本手法が適用可能であるかどうか実験を行う．

6. 今後の課題

今後は本手法を実現するシステムを完成させ，本手法が有用であるかどうか評価実験を行う．現在予測される課題としては，何度も呼び出される下位のモジュールのメソッドに対し，似たようなテストケースが大量に生成されてしまうこと，オブジェクトを多く必要とするメソッドの単体テストプログラムを生成すると，プログラムの長さが膨大になる恐れがあることなどが挙げられる．本研究では，オブジェクトの状態を正確に再現し，自動的に単体テストを生成することを最重視するため，これらの効率面の改善は本研究の範囲とせず，今後の課題とする．

7. おわりに

本稿では，レガシーコードに対し単体テストを書くことは困難であるという問題を取り上げ，その理由として 3 つの問題を挙げた．この問題を解決するための手法として，プログラム全体を動かすテストケースを流し，実行履歴を解析することで単体テストプログラムを自動生成する手法を提案した．この提案により 3 つの問題が解決され，単体テストの自動生成が可能になった．

参考文献

[1] マイケル･C･フェザーズ,レガシーコード改善ガイド,翔泳社,2009
[2] S. Elbaum , H. N. Chin , M. B. Dwyer , M. Jorde, “Carving and Replaying Differential Unit Test Cases from System Test Cases”, IEEE Transactions on Software Engineering, Vol.35 No.1, p.29-45, January 2009
[3] “JUnit”,http://www.junit.org/
[4] 櫻井孝平,増原英彦,古宮誠一,” Traceglasses：欠陥の効率良い発見手法を実現するトレースに基づくデバッガ”，情報処理学会論文誌プログラミング(PRO),3(3),1-17 (2010-06-16) ,1882-7802