Javaを対象としたソースコードインスペクションツールの開発
—テストデータ自動生成ツールの開発—

M2008MM031 山本陽司
指導教員：野呂昌満

1　はじめに

本稿では株式会社キャナリーリサーチ (以下，キャナリーリサーチ) と南山大学との共同研究および OJL の事例報告を行う．本 OJL では PLSE に基づいたモデルからのテストデータの自動生成による JCI[3] のテストプロセス改善について考察し，実現を行った．本 OJL のメタ技術は PLSE，ベース技術はプログラム解析技術 [1] とモデルベースドテスト [2] である．専用手法は JCI 開発におけるテストプロセスの改善である．

JCI は，本 OJL において開発された Java を対象としたソースコードインスペクションツールである．JCI は欠陥として定義したソースコードの構文・制御フロー・データフローのパターンを警告する．JCI は PLSE に基づいて開発されており，キャナリーリサーチからの検査処理に対する要求の変更を仕様に反映，実装し，ソフトウェアテスト (以下，テスト) を実施している．JCI の開発は PLSE に基づき行われており，テストプロセスやテストデータは核資産として管理されている．

テストを完全に行うには，正しいテストを網羅的に行う必要がある．どのようなテストを正しいとするかは，要求によって決まる．要求はドメイン依存するものである．すべての要求の変更を予測する事は不可能なので，その変更に対応して正しいテストを網羅的に行うのは不可能である．したがって，要求の変更に対して柔軟かつ即応的に対応することが求められる．

本 OJL の目的は，JCI の検査処理モデル，テストデータ自動生成ツール (以下，JTDG) を核資産として開発し，利用することで，JCI のテストプロセスを改善することである．検査処理に対する要求は意味・論理 (以下，意味) の単位で変更されるので，意味を用いて欠陥とするソースコードのパターンを表現できると考えた．さらに，その意味の組み合わせのパターンを新たな意味として定義できる．テストデータと対応づけた意味の観点で検査処理を抽象化したモデルを定義し，記述したモデルからテストデータ作成を自動化する．これにより，検査処理に対する要求の変更に対して柔軟かつ即応的に対応することを可能にする．

本 OJL では，PLSE に基づいて検査処理のモデルと JTDG を核資産として開発し，利用することでテストプロセスを改善する．プログラム解析のソースコードをグラフや数値などで表す事で抽象化する性質に基づいて，検査処理対象のソースコードを意味の単位で抽象化する．さらに，モデルベースドテストに基づいて，意味を用いて記述した検査処理モデルからテストの作業を行うアプローチを採用することで，テストプロセスの改善を行う．

2　Javaを対象としたソースコードインスペクションツール (JCI)

2.1　JCI 概要

ソフトウェアの品質を高める方法の 1 つに欠陥の発見と修正がある．ソフトウェアインスペクションは欠陥を発見する手法の 1 つで，ソフトウェア開発における成果物であるドキュメントやソースコードを，対象の作成者以外が，目視によって欠陥を発見，指摘する事である．

JCI は，Java を対象としたソースコードインスペクションツールである．JCI ではソースコードの構文，制御フロー，データフローにおけるパターンを定義し，警告する．JCI は，検査処理の追加・変更が他の検査処理に影響せず，容易に追加・変更できる設計となっている．

2.2　JCI の検査処理に対するソフトウェアテスト

ソフトウェアテストはソフトウェアに求められる要求に対する動作や機能の保証が目的でおこなわれる動的解析の 1 つである．JCI の検査処理への要求に対して個々の検査処理ごとにブラックボックステストを行う．

検査処理には多種多様な要求の変更があり，要求の変更に柔軟かつ即応的に対応してテストを実施することが求められ，頻繁な要求の変更に対してテストを実施することは労力がかかる．特にテストデータ作成ではパターンに当てはまるソースコードを多量に記述する必要があり，労力がかかる．テスト担当者によっては，重複した意味を持つテストデータや，期待する機能を十分にテストできないテストデータを作成する恐れがある．

JCI ではテストをテストデータ作成カタログ [4](以下，カタログ) を利用してテストを行っている．カタログは，検査における着目対象とその分類によって対応するテストデータとなるソースコード片を提供する．カタログを利用すると作成したテストデータを基に部分的な変更で，テストするべきテストデータを作成できるので，要求の変更を反映した検査の仕様に対してどのような基準でテストデータを作成するか決定する労力を削減できる．カタログの例を図 1 に示す．

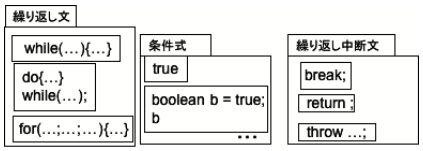

図 1　JCI のテストデータ作成カタログ

しかし，カタログでは要求の変更に対して仕様を作成し，その仕様からテスト担当者が手動でテストデータを作成

するので，要求の変更に対して柔軟かつ即応的に対応することは困難である．現在のテストプロセスを図 2 に示す．

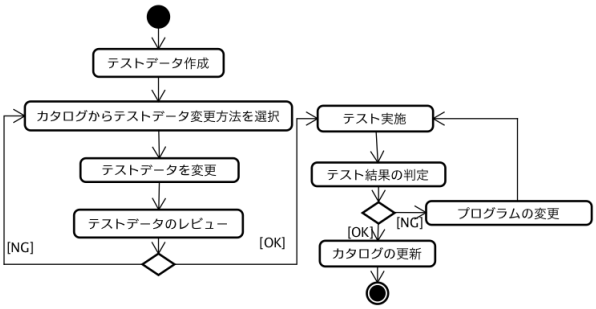

図 2 JCI のテストプロセス

3 モデルベースドテストの適用

モデルベースドテストはソフトウェアテスト手法の 1 つで, テスト対象の特定の性質を表現したモデルを基にテストケースの作成や，テスト結果の評価などの一般的なテストの作業を行うアプローチである．
JCI はプログラム解析を利用して，ソースコードを抽象構文木・制御フロー・データフローと捉えてそのパターンを欠陥として警告する．検査処理の要求は意味の単位で変更されるので，検査対象のソースコードを意味の組み合わせで捉えることできると考えた．モデルベースドテストに基づいて意味の組み合わせを記述したモデルを定義し, 構文やフローのパターンとの対応関係を明らかにする事で, 要求の変更に柔軟かつ即応的に対応し，その対応関係を基にテストデータを自動生成することでテストの省力化ができると考えた．

3.1 検査処理の分析

実際に作成した全ての検査処理の分析を行った結果，作成した検査処理は特定の意味を表す要素への着目，値への着目，値の判定を組み合わせて，新たに特定の意味を表す要素に着目し，その組み合わせのパターンを警告するという流れで表す事ができる事が分かった．
分析して定義した検査処理の抽象モデルとして検査処理の流れと構成を図 3 に示す．

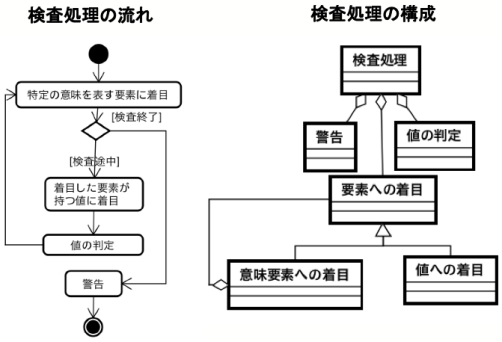

図 3 検査処理の流れと構成

3.2 JCI の検査処理に適したモデルの選択と記述

無限ループを警告する検査処理では，以下の条件に当てはまる繰り返し文を警告する．

- 繰り返し文で条件式内の変数が繰り返し文内で一度も更新されていない
- ループを中断する文が存在するものは警告しない

無限ループを警告する検査処理では，検査対象に存在する for 文，while 文，do-while 文に着目する．for 文，while 文，do-while 文の条件式内に変数参照が存在しない場合は，その繰り返し文への着目をやめて，次の繰り返し文に着目している．このように，特定の構文に着目したり，特定の値を持つ構文要素のみに着目することを，1 つの意味と捉えられると考えた．
for 文，while 文，do-while 文を条件式・繰り返す文を持つ「繰り返し文」という意味で捉えることで，for 文，while 文，do-while 文という構文を意識する事なく，条件式や繰り返す文に着目できる．さらに着目した繰り返し文に対して，条件式内に変数参照が存在するものにのみ着目する場合は,「条件式内に変数参照が存在する繰り返し文」という意味で捉えることができる．このように構文やフローのパターンを意味として定義することで，欠陥となるソースコードを意味の組み合わせで表すことができる．分析した検査処理の抽象モデルと定義した意味に基づいて無限ループを警告する検査処理を記述すると，図 4 のように表すことができる．

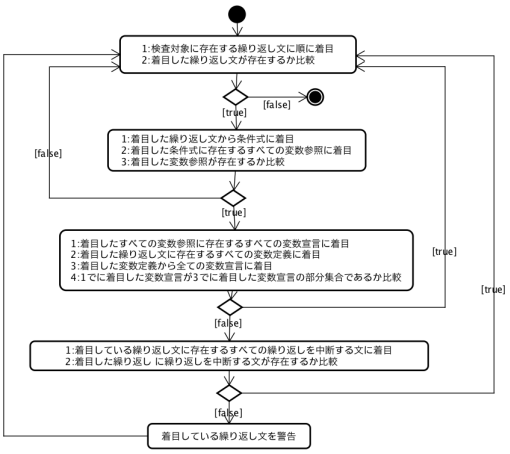

図 4 無限ループを警告する検査処理のモデル

3.3 モデルとテストデータの対応関係

提案した検査処理モデルとテストデータを対応付ける．検査処理のアクティビティである特定の意味を表す要素への着目，値への着目は特定のソースコード片と対応づける事ができ，値の判定とその結果によってソースコード片を組み合わせる事ができる，要求に対して作成した検査処理モデルに基づいて開始から終了までのパスに対するソースコードの組み合わせをテストデータとして網羅的に作成する事で，要求に対して柔軟かつ即応的に対応する事ができる．

例えば，検査対象に内に存在する順に繰り返し文が存在して条件式に変数参照が存在しないパスを考えた場合，繰り返し文である for 文，while 文，do-while 文を記述したソースコード片と，変数参照を持たない boolean 型の式を記述したソースコード片を組み合わせてパスに対応するテストデータとなるソースコードを作成する．他の例として，また，検査対象に内に存在する順に繰り返し文が存在して条件式に変数参照が存在し，繰り返す文にその変数参照に対する定義点がある場合は，繰り返し文である for 文，while 文，do-while 文を記述したソースコード片と，変数参照を持つ boolean 型の式を記述したソースコード片に加えて，変数参照に対する定義点を記述したソースコードを組み合わせて，パスに対応するテストデータとなるソースコードを作成する．無限ループを警告する検査処理のモデルに対するテストデータの例を図 5 に示す．

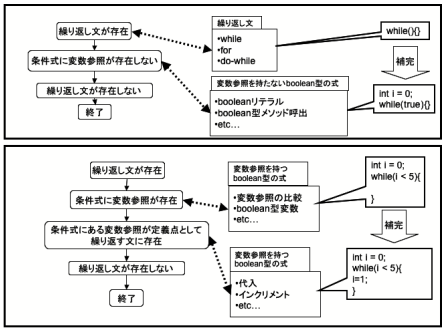

図 5 無限ループを警告する検査処理のモデルに対するテストデータの例

4 JCI テストデータ自動生成ツール (JTDG)

4.1 テストデータ自動生成ツールの概要

JTDG は JCI の検査処理のもでるを入力として，テストデータであるソースコードを出力するツールである．検査モデルの開始から終了までのパス上の情報からテストデータを自動生成する．JTDG を利用することで，要求の変更に柔軟かつ即応的に対応したテストデータ作成を行い，省力化することができる．

検査処理のモデルは検査処理への要求を基に作成する．JTDG に入力したモデルから自動生成されたソースコードを対象の検査処理に入力し，その結果と期待結果を比較する．全て合格した場合は，テストを終了し，不合格なテストデータに対しては，不合格の原因を調査し，原因となったモデル，検査処理を修正し，再テストを行う．

4.2 テストデータ自動生成ツールの設計

JTDG を実現するためには，検査処理モデルの解析，コードの生成が必要である．JCI ではソースコードを構文として捉えるために，抽象構文木を作成している．この抽象構文木を再利用し，検査処理モデルの着目する意味や，値の判定の種類毎に対応する抽象構文木に変換し，抽象構文木からテストデータとなるソースコードを生成できると考えた．JTDG の設計を図 6 に示す．

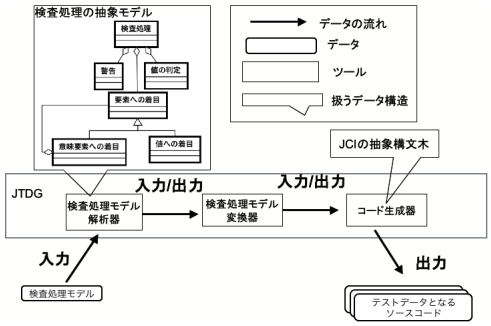

図 6 JTDG の設計

5 テストプロセスの改善

現在のテストプロセスを基に検査処理モデルと JTDG を利用したテストプロセスを定義した．定義したテストプロセスを図 7 に示す．

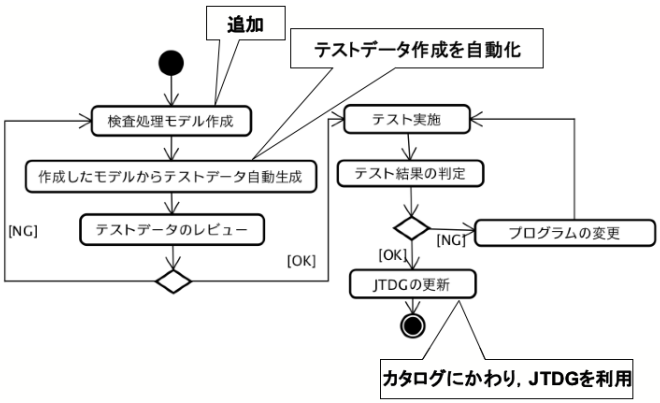

図 7 再定義した JCI のテストプロセス

JTDG の検査処理に対する要求をを満たして記述した検査処理モデルから対応するテストデータを生成することで，要求の変更に対して，柔軟かつ即応的に対応することができる．また，テストデータは自動生成可能なので，テストデータ作成を省力化することができる．

6 考察

JTDG の試作を実現し，いくつかの検査処理に対する要求の変更に対して提案したテストプロセスに基づいてテストを行った結果を考察した．

6.1 検査処理モデルに関する考察

提案する検査処理モデルを用いたテストで要求の変更に対して柔軟かつ即応的に対応できるか考察する．無限ループを警告する検査処理に繰り返し文の条件式が true であり，繰り返す文に繰り返しを中断する文も警告したいという要求があった場合，無限ループを警告する検査処理は図 8 の様に記述できる．

この要求に対して検査処理モデルを変更したのは，条件式が true リテラルかの判定のみである．それ以外のモデルは全て再利用する事ができる．モデルの変更部である条件式が true リテラルかどうかの比較に対するテストデー

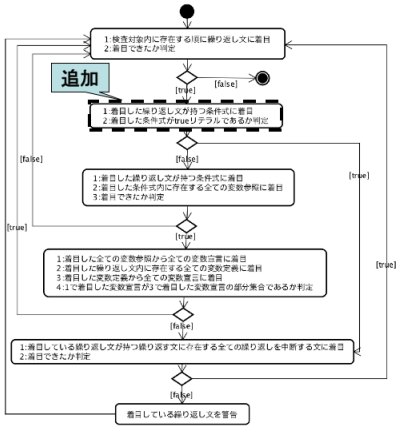

図 8 要求の変更に対応した無限ループを警告する検査処理モデル

タを作成することで，新たな要求に対して柔軟かつ即応的に対応することができる．

6.2 テストデータの自動生成に関する考察

作成した検査処理モデルの部品を利用可能な他の検査処理のモデルを記述し，再利用した部分に対応するソースコードが．その検査処理に正しいテストデータであるか考察する．
繰り返し文を中断する箇所が複数ある場合に，その繰り返しを中断する文を警告する検査処理は図 9 の様に表す事ができる．

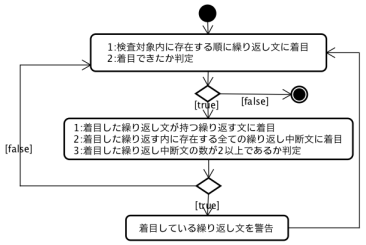

図 9 複数の繰り返し中断箇所を警告する検査処理モデル

この検査では無限ループを警告する検査処理と同じように繰り返し文に着目しており，無限ループを警告する検査処理と同じように while 文，for 文，do-while 文のテストデータを作成する必要がある．また，繰り返し文を中断する文についても同様で無限ループを警告する検査処理と同じようにテストデータを作る必要がある．
図 9 のようにモデルを記述すれば，同じ部品を使う事となり，再利用した部分は対応する同じソースコードが自動生成される．具体的なテストデータを意識する事なく，検査処理をモデルで記述するのみで，作成する必要があるテストデータを自動生成する事ができる．

6.3 定義したテストプロセスに関する考察

現在のテストプロセスと本稿で定義したテストプロセスの要求に対する変更を比較することで，定義したテストプロセスが要求の変更に対して柔軟かつ即応的に対応できるか考察を行う．
現在のテストプロセスで要求の変更があった場合には，要求に対して仕様を定義する必要がある．仕様は要求の変更単位である意味と，構文のパターンと対応づける必要があり，柔軟かつ即応的に対応してテストを行うのは困難である．本稿で定義したテストプロセスでは要求の変更に対してその変更の単位である意味で検査処理モデルを記述することができ，対応する構文のパターンと対応づいているので，現在のテストプロセスに比べて，要求の変更に対して柔軟かつ即応的に対応してテストを行うことができる．
また，現在のテストプロセスではテストデータの作成はテスト担当者が手動で行っている．テストデータであるソースコードを記述するのは，仕様で定義した構文のパターンを実現する必要があり，労力がかかる．定義したテストプロセスでは作成した検査処理モデルからテストデータであるソースコードを自動生成するので，現在のテストプロセスに比べて，少ない労力でテストを行うことができる．

7 おわりに

本稿は，JCI 開発に対するテストプロセスの改善をおこなった．テストプロセスの改善では，検査処理モデルと，JTDG を利用したテストプロセスを定義した．定義したテストプロセスに対しての妥当性の考察をおこなった．
本稿のメタ技術は JCI のテストプロセスの改善に用いた PLSE である．ベース技術は検査対象であるソースコードを意味として捉えたプログラム解析技術，テストデータの自動生成によるテストプロセス定義のために利用したモデルベースドテストである．
今後の課題は，JTDG を実現することである．また，検査処理モデル，JTDG を利用したテストプロセスを用いて実際の要求に対応し，提案したテストプロセスが妥当であることを示す必要がある．

参考文献

[1] D.Jackson, M. Rinard, “Software Analysis: A Roadmap,” *The Future of Software Engineering* ,pp.135-145, 2000.

[2] I. K. El-Far, J.A. Whittaker, “Model-based software testing,” *In Encyclopedia on Software Engineering(edited by J.J. Marciniak)* , Wiley. 2001.

[3] 浦野彰彦, 沢田篤史, 野呂昌満, 蜂巣吉成, “デザインパターンを用いたソースコードインスペクションツールのソフトウェアアーキテクチャ設計,” ソフトウェア工学の基礎 XVII (日本ソフトウェア科学会 FOSE2010), 近代科学社, 2010, pp. 15-24.

[4] 二宮剛史, “ Java ソースコードの CDI(Code Inspection) ツールの開発 —コードインスペクションツールに対するテストプロセスの改善—, ”南山大学大学院数理情報研究科 2009 年度 修士論文 (OJL 成果報告書), 2010.