JDBC プログラムの単体テストにおける例外処理部の優先順位付け

Priorization of JDBC Exception Handling for Unit Testing

藤本 恭平 † 福安 直樹 ‡ 満田 成紀 ‡ 松延 拓生 ‡ 鯵坂 恒夫 ‡
Kyohei Fujimoto Naoki Fukuyasu Naruki Mitsuda Takuo Matsunobe Tsuneo Ajisaka

1. はじめに

ソフトウェアをテストする目的は，欠陥を取り除いて，ユーザの要求を満たす，品質の良いソフトウェアを作ることである [1]．すべてのコードをテストしなければ，品質の低下とバグ発見時の手戻り作業による工数増加をまねき，プロジェクトのコストが増したり，納期が遅れるなどの問題が生じる恐れがある [2]．

しかし，一般にプログラムが複雑になるにつれてテストを通しにくい箇所が増えるため，テストの網羅率 100%を実現することは難しくなる．未テストコードとなりやすい例として，テストケースの漏れ，デッドコード，例外処理がある [2]．

このうち，例外処理はプログラムの動作中に異常な事象（例外）が発生した際に実行される処理であり，通常の処理と制御の流れが異なるため，動作の解析が困難である [3] ことから，完全なテストが難しい．そのような場合に重要度の高い例外処理を優先的にテストすることが必要になると考え，本論文では，JDBC プログラムの単体テストにおける例外処理部の優先順位を判断する手法を提案する．

2. JDBC プログラムにおける例外処理

JDBC (Java Database Connectivity) API は，Java 言語とデータベース間でデータベース非依存の接続を行うための業界標準である [4]．プログラムの実装にあたっては，データベースの安定性を維持するために例外の対処を考慮しておくことが必須となる．

まずは，JDBC プログラム中に記述される例外処理について，事例集に基づき分類を試みる．

2.1 調査の対象と手順

ソースコード集『JDBC Recipes[5]』（コード集 A と呼ぶ）及び『Expert Oracle JDBC Programming[6]』（コード集 B）に含まれる計 326 節の catch 節を調査する．いずれも JDBC によるデータベースの運用に関わる一式のレシピを，コピーアンドペーストで使うことができるようにソースコードの具体例が記載されている書籍である．

表 1: catch 節で捕捉する例外クラスの内訳

例外クラス	コード集 A	コード集 B	計
Exception	129	15	144
SQLException	119	42	161
ClassNotFoundException	7	0	7
BatchupdateException	6	0	6
ParseException	4	0	4
ArithmeticException	1	0	1
IOException	1	0	1
NoSuchAlgorithmException	1	0	1
UnknownHostException	1	0	1
計	269	57	326

ソースコード集に記述された catch 節で捕捉している例外クラスの内訳を表 1 に示す．JDBC の取り扱い説明を目的とする書籍であるので，一般的な例外クラスである Exception クラスと，おもにデータベースのアクセスエラーに関する例外クラスである SQLException クラスが大多数を占めている．そこで，調査では Exception クラスまたは SQLException クラスの例外を補足する catch 節を対象とする．catch 節中に記述された命令文を集計し，その処理内容を抽象化して分類する．

2.2 catch 節中の命令文の分類

調査対象の catch 節中に記述された命令文を，変数名のみ異なる例は同じ命令文として集計すると 28 通りあった．その処理内容を抽象化し分類した結果が表 2 で，小分類 9 項目，大分類 4 項目に分類できた．「表示や記録」を行う命令文が 209 文と最も多く，次に「終了」が 75 文，「回復」が 21 文と続いた．

表 2 の小分類の各項目は以下の通りである．

スタックトレースを表示

`printStackTrace` メソッドでスタックトレースと例外クラス，例外の詳細メッセージなどを表示する処理である．Java の実行中は，メソッドが呼び出される度にそのメソッド名がスタックに記録される．スタックトレースはその記録から，例外の発生した行番号と通過したメソッド名とメソッドの呼び出し

† 和歌山大学大学院システム工学研究科, Graduate School of Systems Engineering, Wakayama University
‡ 和歌山大学システム工学部, Faculty of Systems Engineering, Wakayama University

表 2: catch 節中に見られる命令文の処理内容による分類

大分類	小分類	該当
表示や記録	1. スタックトレースを表示	128
	2. エラーメッセージを表示	75
	3. ログを記録	6
回復	4. トランザクションをロールバック	21
終了	5. メソッドを終了 (return)	9
	6. システムを終了 (exit(1))	66
その他	7. 例外を変換して投げ直し	8
	8. 自動コミットを有効化	3
	9. 例外を無視	19
計		335

箇所を順に遡って表示する．この情報を手掛かりとして，問題のある箇所が特定できる．

エラーメッセージを表示

上記のスタックトレースは表示せず，補足された例外クラスと例外の詳細メッセージを表示する処理である．

ログを記録

発生した例外の例外クラスをログに記録する処理である．

トランザクションをロールバック

データベース管理システムのロールバック機能を呼び出して，トランザクションが失敗する前の状態に巻き戻す．try 節でデータ更新を行う際に catch 節でロールバックを行うようにしておけば，トランザクションの原子性を保証し，データベースの整合性を保つことができる．

メソッドを終了 (return)

return 文でメソッドを終了させ，呼び出し元に処理を戻す処理である．

システムを終了 (exit(1))

exit メソッドを呼び出してシステムを終了させる処理である．

例外を変換して投げ直し

throw 文で捕捉した例外を明示的に投げ直す処理である．

自動コミットを有効化

try 節中でデータベースの更新前に，setAutoCommit(false) で自動コミットを無効化し，更新後に，setAutoCommit(true) で自動コミットを有効化する処理を記述しているため，データ更新において例外が発生し，残りの処理が行われない時にも同様に自動コミットを有効化する処理である．Java の初期設定では自動コミットが有効化されており，データ更新した直後に自動的にコミットされ，トランザクションが確定する．しかし，更新作業で例外が発生した場合，例外処理によって対処できない．そのため，例外発生の可能性があるデータ更新ではあらかじめ自動コミットを無効化しておく必要がある．

例外を無視

try-catch 節を記述するものの，対処する必要がない場合や，対処の手段がない場合に意図的に空の catch 節を記述し，例外処理を行わない場合がある．

2.3 分類の考察

調査結果から JDBC プログラムにおける例外処理には,「表示や記録」,「回復」,「終了」,「その他」の処理が含まれていることが分かった．これらの処理の目的は,「表示や記録」は例外の情報をユーザまたは開発者に提供すること,「回復」は例外発生前の正常な状態に回復させること,「終了」は処理の継続を放棄して例外が発生した処理やシステムを終了させることである．

このうち,「表示や記録」は例外への対処をユーザに託すために情報を提示するに留まる処理である．これに対して,「表示や記録」以外の処理は開発者が想定した対処をプログラム側で行う処理である．すなわち，JDBC プログラムにおける例外処理は，例外からの回復可能性の有無によって二分できる．

3. 優先順位付け手法

例外処理の重要性を比較できるようにするため，次の手順を取る．まず，重要度の決定要因を複数検討する．次に，各要因について想定される状態を設定し，その状態に水準を与える．最後に，重要度の表現方法を考案する．

なお，ここで提案する要因，水準については実際の開発方針に応じて適宜変更できる．

3.1 重要度決定に必要な要因

例外処理の重要度決定に必要な要因を探す．ここで採用する要因は，本論文の手法を用いる開発者が，各要因の水準を決定できるものである必要がある．そのため，JDBC プログラムにおいて一般的な要因を挙げることを目標とする．

まず，バグ管理においてバグの重要度を決定する既存の手法を参考にした．例外処理は管理者が異常な動作を正常系の動作の範囲内として対処するのに対して，バグ管理はプログラムから取り除かれるべき誤りを除去する点が異なっているが，いずれも通常の処理を妨げる事象

である点は同じである．そこで，両者の重要度決定に必要な要因は重複していると判断した．また，バグ管理においてバグ修正の優先順位を 5 段階で定義している事例 [7] があり，テストの優先順位についても同様に定義することができる考えた．

栗栖 [8] によると，バグ管理のノウハウとして，バグの重要度決定に必要な項目には「影響範囲」,「データ更新の有無」,「発生頻度」,「関連バグの数」がある．これを例外処理に当てはめると,「影響範囲」,「データ更新の有無」,「発生頻度」はそのまま使用できると考える.「関連バグの数」は「関係する処理の数」と置き換える．

次に，バグ管理にはなく，例外処理では考える必要のある要因を探す．第 2 章の考察から「回復可能性」の有無が例外処理の重要性に関わると考えられるため，これを追加する．

さらに，直観的に水準を指定できるように要因を分割する.「影響範囲」を「影響範囲 A（影響を受けるシステムの範囲)」と「影響範囲 B（影響を受けるユーザの範囲)」に,「データ更新」を「データ更新 A（try 節)」と「データ更新 B（catch 節)」に,「発生頻度」を「発生頻度 A（例外の発生確率)」と「発生頻度 B（try 節の利用頻度)」にそれぞれ分割する．

以下に，各要因の説明を行う．各要因のラベルを F1 〜F5 とし，分割した要因には a, b を付す．

F1a: 影響範囲 A（影響を受けるシステムの範囲）
テスト対象の例外処理によって補足する例外がシステム中のどのくらいの範囲に影響を与えるかを指定する．

F1b: 影響範囲 B（影響を受けるユーザの範囲）
テスト対象の例外処理によって捕捉する例外が誰に影響を与えるかを指定する．

F2: 関係する処理
テスト対象の例外処理が含まれるメソッドと関係する処理の数を指定する．例えば，UML のクラス図において，テスト対象のクラスと線で結ばれるクラスが多いほど，関係する処理が多いと考える．

F3a: データ更新 A（try 節）
テスト対象の例外処理における try 節中の，データベースの更新を行う命令文の有無（データベース管理における基本操作である CRUD (Create, Read, Update, Delete) のうち Create, Update, Delete の有無）を指定する．ある場合，データ更新の量の大小も指定する．

F3b: データ更新 B（catch 節）
F3a と同じように，catch 節についてもデータ更新の有無を指定する．

F4a: 発生頻度 A（例外の発生確率）
テスト対象の例外処理によって捕捉する例外の発生確率を指定する．

F4b: 発生頻度 B（try 節の利用頻度）
テスト対象の例外処理の含まれるメソッドの利用頻度を指定する．

F5: 回復可能性
テスト対象の例外処理によって異常から回復できる可能性があるかを指定する．具体的には，失敗したデータベースアクセスに対してロールバック処理等を行うことによって異常発生前の状態に巻き戻せる可能性があるかなどによって判断する．表 2 の分類における「表示や記録」のみを行う例外処理では回復可能性は無いと判断できる．

3.2 各要因の水準設定と寄与度の仮定

前節で挙げた各要因について考えられる状態を表 3 のように設定する．各要因の状態は 1 が最も重要度が高く，1 → 2 → 3 →……の順で重要度が低下する．また，設定した状態を「高」,「中」,「低」の 3 水準（要因によっては「高」,「低」の 2 水準）に分けた．

3.3 重要度の決定

テスト対象の例外処理を処理 A とする．8 要因について前節で設定した表 3 の水準から処理 A に当てはまる状態をそれぞれ選ぶ．この時，重要度の高い状態から順に見ていき（1 → 2 → 3 →……の順），処理 A に初めに当てはまったものを処理 A の状態とする．選んだ状態に該当する水準（「高」,「中」,「低」のいずれか）を記録する．これによって処理 A の 8 つの水準を決定する．「高」,「中」,「低」の数を集計したものを処理 A の重要度とする．

3.4 優先順位付け

テスト対象となる複数の例外処理について，前節の手法で指定した重要度を比較する．初めに「高」の数を比較し,「高」の数が多い処理ほど重要度が高くテストの優先順位が高い例外処理と考えられる.「高」の数が同じ場合,「中」の数で比較する．

4. 手法の適用

考案した優先度順位付け手法の適用事例を示す．第 2 章の調査対象及びオープンソースの JDBC プログラムに含まれる try-catch 節に適用して重要度の比較を行い，どちらの例外処理を優先してテストすべきかを判断する．紙面の都合上，対象とするソースコードと適用の手順は前者の例のみ記載する．ここでは 2 つの処理を比較しているが，3 つ以上の処理を同時に比較してもよい．

表 3: 各要因の水準

ラベル	要因	水準	状態
F1a	影響範囲 A （影響を受けるシステムの範囲）	高	1. システム全体に影響がある 2. システムの広範囲に影響がある 3. システムに影響がある
		中	4. 2 機能あるいは 2 画面以上に影響がある
		低	5. 1 機能あるいは 1 画面以内に影響がある 6. 影響がない
F1b	影響範囲 B （影響を受けるユーザの範囲）	高	1. エンドユーザに影響がある
		中	2. 開発者に影響がある
		低	3. 影響がない
F2	関係する処理	高	1. 関係する処理が多くある
		中	2. 関係する処理がある
		低	3. 関係する処理がない
F3a	データ更新 A （try 節）	高	1. データ更新が多くある
		中	2. データ更新がある
		低	3. データ更新がない
F3b	データ更新 B （catch 節）	高	1. データ更新が多くある
		中	2. データ更新がある
		低	3. データ更新がない
F4a	発生頻度 A （例外の発生確率）	高	1. 通常の利用で頻繁に発生する
		中	2. 通常の利用で発生する 3. 特定条件下で頻繁に発生する
		低	4. 特定条件下で発生する 5. 発生しない
F4b	発生頻度 B （try 節の利用頻度）	高	1. 頻繁に利用される
		中	2. 時々利用される
		低	3. 稀に利用される 4. 利用されない
F5	回復可能性	高	1. 回復可能性がある
		低	2. 回復可能性がない

4.1 適用事例 1

図 1 のプログラム * は，表を作成しデータを挿入する処理である．catch 節には，エラーメッセージの表示とトランザクションのロールバック，自動コミットの有効化を行う命令文が記述されている．表 2 の大分類における「表示や記録」,「回復」,「その他」の処理が行われる．try 節では，データベースへ接続，自動コミットを無効化，データベースに新規の表を作成，作成した表にデータを追加，トランザクションをコミット，自動コミットを有効化という順序で命令文が記述されている．

各要因について順に見ていくと,「F1a: 影響範囲 A（影響を受けるシステムの範囲）」と「F1b: 影響範囲 B（影響を受けるユーザの範囲）」は対象がソースコード集であるため，システムやユーザを考慮できないので，ここでは選択できない．try 節で他のメソッドを呼び出したりメンバ変数に変更を加えたりはしていないので「F2: 関係する処理」は「3. 関係する処理がない」を選択する．try 節中のデータ更新は，表作成とデータ追加の 2 つがあるが，オーダーは小さいので「F3a: データ更新 A（try 節）」は「2. データ更新がある」を選択する．catch 節中のデータ更新はロールバックがあるが，同様にそのオーダーは小さいので「F3b: データ更新 B（catch 節）」は「2. データ更新がある」を選択する．try 節でユーザ

*Mahmoud Parsian 著, “ JDBC Recipes ”, Apress, 2005, p.44.

```
1    try {
2        conn = DriverManager.getConnection(dbURL, dbUser, dbPassword);
3        conn.setAutoCommit(false);
4        stmt.executeUpdate("CREATE TABLE cats_tricks(" + "name VARCHAR(30), trick VARCHAR(30))");
5        stmt.executeUpdate("INSERT INTO cats_tricks(name, trick) " + "VALUES('mono', 'rollover')");
6        conn.commit();
7        conn.setAutoCommit(true);
8    } catch(SQLException e) {
9        System.out.println("SQL Exception/Error:");
10       System.out.println("error message=" + e.getMessage());
11       System.out.println("SQL State= " + e.getSQLState());
12       System.out.println("Vendor Error Code= " + e.getErrorCode());
13       conn.rollback();
14       conn.setAutoCommit(true);
15   }
```

図 1: 表を作成しデータを挿入する処理

```
1    try {
2        conn = ...
3        conn.setAutoCommit(false);
4        stmt = conn.createStatement();
5        stmt.executeUpdate("create table dept_table(...)");
6        String insert = "insert into dept_table values (?, ?)";
7        pstmt = conn.prepareStatement(insert);
8
9        pstmt.setString (1, "sales");
10       pstmt.setString (2, "Troy");
11       pstmt.addBatch();
12
13       pstmt.setString (1, "business");
14       pstmt.setString (2, "Los Angeles");
15       pstmt.addBatch();
16
17       // データをさらに複数追加できる
18
19       int[] updateCounts = pstmt.executeBatch();
20       rs = stmt.executeQuery("select * from dept_table");
21       while (rs.next()) { ... }
22       conn.commit();
23   } catch(Exception e) {
24       e.printStackTrace();
25       conn.rollback();
26   }
```

図 2: 表を作成しバッチ処理でデータを追加する処理

の入力を受けたり，複雑な処理をしたりはしていないので通常の運用下で例外が発生することは考えにくいため「F4a: 発生頻度 A（例外の発生確率)」は「4. 特定条件下で発生する」を選択する．try 節では汎用性の低い処理をしているので「F4b: 発生頻度 B（try 節の利用頻度)」は「3. 稀に利用される」を選択する．catch 節でエラーメッセージの表示や後処理だけでなく，ロールバック処理を行っているので「F5: 回復可能性」は「1. 回復可能性がある」を選択する．

以上のようにして選択した状態を，該当する「高」,「中」,「低」のいずれかの水準に変換して表 4 の処理 1（図 1）の行に記録した．

図 2 のプログラム † は表を作成しバッチ処理でデータを追加する処理である．適用の手順は省略するが，図 1 の処理と同じように該当する水準を選択し，表 4 の処理 2（図 2）の行に記録した．特筆すべき点は，バッチ処理によって溜め込んだ SQL 文を一括してデータベースへ

†Mahmoud Parsian 著, “ JDBC Recipes ”, Apress, 2005, p.62.

表 4: 適用事例 1

テスト対象	水準								水準の数			優先順位
	F1a	F1b	F2	F3a	F3b	F4a	F4b	F5	高	中	低	
処理 1（図 1）	-	-	低	中	中	低	低	高	1	2	3	2
処理 2（図 2）	-	-	低	高	高	中	高	高	4	1	1	1

表 5: 適用事例 2

テスト対象	水準								水準の数			優先順位
	F1a	F1b	F2	F3a	F3b	F4a	F4b	F5	高	中	低	
処理 o1	低	高	中	高	低	低	高	低	3	1	4	1
処理 o2	高	高	中	低	低	中	高	低	3	2	3	2

渡している（19 行目）点である．preparedStatement() メソッドは，複数のデータを追加する時に，変動する箇所に「?」を埋め込んだ SQL 文をあらかじめコンパイルしてデータベースに渡しておく（6 行目）ことで，処理を効率化する．すなわち，大きなオーダーのデータ更新が行われるため，F3a，F3b を「高」とした．

4.2 適用事例 2

前節と同様に Java 言語で書かれたオープンソースの JDBC プログラム「SQLeonardo[‡]」に含まれる try-catch 節について適用した.「SQLeonardo」は SQL を理解せずに GUI 上でデータベースの操作ができるツールである．

表のデータを全て削除する処理（処理 o1）と表から取り出したデータを返す処理（処理 o2）について適用した結果が表 5 である．前節の適用例と異なり完成されたソフトウェアであるため,「F2: 関係する処理」については，メソッドの呼び出し元が何箇所あるかを考慮した．

4.3 考察

適用事例 1 では，処理 2（図 2）のほうが「高」の水準が圧倒的に多いため，処理の重要度が高く，テストの優先順位が高いと考えることができる．

適用事例 2 では，水準の数に大きな差が見られず，優先順位は断定し難い．しかし，各要因に着目すると F1a や F3a で大きな差が見られるので，その要因ごとに比較して優先順位を判断する手段が考えられる．F1a では「影響を受けるシステムの範囲」が指定されており，この水準が高い例外処理のテストを残したままソフトウェアをリリースするのは危険であると判断する場合は，この水準が高いほうのテストを優先すべきである．F3a では「try 節におけるデータ更新の量」が指定されているが，更新するデータの重要性を考慮して，この要因が重要であるかを判断する必要がある．このような場合はプロジェクトの方針に依存するので，単に水準の数で重要性を判断することはできない．よって，この手法は優先順位を一意に決めるよりも，各要因に観点を絞って 2 つ以上の例外処理の重要性を比較できることに意義があると考えられる．最終的に優先順位の判断は開発者に委ねられるが，事例研究によって各要因の寄与度を統計的に算出できれば，判断材料としての信頼性を向上させることは可能であると考える．

また，提案した手法はテスト計画を立てる際に利用することを想定しているが，適用例では既存の例外処理を対象としている．そのため，実際にこのテスト計画でテストを進めると有効なテストができるのかという観点の考察は出来ていない．

5. おわりに

事例集の調査から，JDBC プログラムにおける例外処理の catch 節に含まれる命令文は，処理内容について「表示や記録」,「回復」,「終了」,「その他」に分類できた．回復可能性のある処理とない処理として分類できることが分かったので，これと既存のバグ管理の手法をもとに例外処理の重要度を決定する 8 つの要因を検討した．その要因に水準設定を行い，開発者が手軽に入力できる形式の優先順位付け手法を考案した．考案した手法を適用するとテストの優先順位を判断する一つの指標は得られた．

ただし，この手法は処理の重要度から優先順位を判断するものであり，テストの実施順序を決定するものではない．実際のプロジェクトでは，処理の重要度に加えて，テストに掛かる工数や費用などのコストも考慮したうえで判断する必要がある．

[‡]“ SQLeonardo ”, https://sourceforge.net/projects/sqleonardo/

参考文献

[1] 石原一宏, 田中英和 著, “ ソフトウェアテストの教科書 ”, ソフトバンククリエイティブ, 2012.

[2] JaSST'06 in Tokyo, 2006-01-30/31, “ .NET 開発でカバレッジ 100%を実現 ”, 日本コンピュウェア株式会社 営業技術本部, http://jasst.jp/archives/jasst06e/pdf/E3-2.pdf, (参照 2015-07-23).

[3] 浜口毅, 酒井正彦, 馬場正貴, 阿草清滋 著, “ 例外処理を持つ関数型プログラムの停止性・非停止性証明法 ”, 名古屋大学, 2011.

[4] “ Java SE Technologies - Database ”, Oracle, http://www.oracle.com/technetwork/java/javase/jdbc/index.html, (参照 2015-07-23).

[5] Mahmoud Parsian 著, “ JDBC Recipes ”, Apress, 2005.

[6] R.M.Memon 著, “ Expert Oracle JDBC Programming ”, Apress, 2005.

[7] David Baron 著, “ Bug priorities ”, David Baron's Weblog, 2009-01-20, http://dbaron.org/log/20090120-bug-priorities, (参照 2015-07-23).

[8] 栗栖義臣 著, “【バグ管理の作法】バグ管理のノウハウ ”, Think IT, 2007-12-04/25(隔週), http://thinkit.co.jp/free/article/0712/2/, (参照 2015-07-23).