修士論文

ソースコードの類似性に基づいた テストコード自動推薦ツール

倉地 亮介

2020 年 1 月 28 日

奈良先端科学技術大学院大学
先端科学技術研究科 情報科学領域

本論文は奈良先端科学技術大学院大学先端科学技術研究科情報科学領域に
修士 (工学) 授与の要件として提出した修士論文である。

倉地 亮介

審査委員：

飯田 元 教授	（主指導教員）
井上 美智子 教授	（副指導教員）
市川 昊平 准教授	（副指導教員）
崔 恩瀞 准教授	（京都工芸繊維大学）

ソースコードの類似性に基づいた テストコード自動推薦ツール*

倉地 亮介

内容梗概

ソフトウェアの品質確保の要と言えるソフトウェアテストを支援することは重要である．これまでにテスト作成コストを削減するために様々な自動生成技術が提案されてきた．しかし，既存ツールによって自動生成されたテストコードはテスト対象コードの作成経緯や意図に基づいて生成されていないという性質から後のメンテナンス活動を困難にさせる課題がある．この課題の解決方法として，既存テストの再利用が有効であると考えられる．本研究ではOSSプロジェクト上に存在する既存の品質が高いテストコードを推薦するツールSuiteRecを提案する．SuiteRecは，類似コード検索ツールを用いてクローンペア間でのテスト再利用を考える．開発者からの入力コードに対して類似コードを検出し，その類似コードに対応するテストスイートを開発者に推薦する．さらに，テストコードの良くない実装を表す指標を示すテストスメルを開発者に提示し，より品質の高いテストスイートを推薦できるように推薦順位を並び替える．提案ツールの評価では，被験者によってSuiteRecを使用した場合とそうでない場合でテストコードの作成してもらい，テスト作成をどの程度支援できるかを定量的および定性的に評価した．その結果，SuiteRecを利用した場合，(1) 条件分岐が多いプログラムのテストコードを作成する際にコードカバレッジの向上に効果的であること，(2) 作成したテストコードはテストスメルの数が少なく品質が高いこと，(3) 開発者はテストの作成を容易だと認識し，自身で作成したテストコードに自信が持てることが分かった．

*奈良先端科学技術大学院大学 先端科学技術研究科 情報科学領域 修士論文, 2020年1月28日.

キーワード

類似コード検出, 推薦システム, ソフトウェアテスト, 単体テスト

Automatic Test Suite Recommendation System based on Code Clone Detection*

Ryosuke Kurachi

Abstract

Automatically generated tests tend to be less read-able and maintainable since they often do not consider thelatent objective of the target code. Reusing existing tests mighthelp address this problem. To this end, we present SuiteRec, asystem that recommends reusable test suites based on code clonedetection. Given a java method, SuiteRec searches for its codeclones from a code base collected from open-source projects,and then recommends test suites of the clones. It also providesthe ranking of the recommended test suites computed basedon the similarity between the input code and the cloned code.We evaluate SuiteRec with a human study of ten students. The results indicate that SuiteRec successfully recommends reusabletest suites.

Keywords:

clone detection, recommendation system, software testing, unit test

*Master's Thesis, Division of Information Science, Graduate School of Science and Technology, Nara Institute of Science and Technology, January 28, 2020.

目 次

図 目 次

表 目 次

1. はじめに

近年，ソフトウェアに求められる要件が高度化・多様化する一方，ユーザからはソフトウェアの品質確保やコスト削減に対する要求も増加している [1]．その中でもソフトウェア開発全体のコストに占める割合が大きく，品質確保の要ともいえるソフトウェアテストを支援する技術への関心が高まっている [?]．しかし，現状ではテスト作成作業の大部分が人手で行われており，多くのテストを作成しようとするとそれに比例してコストも増加してしまう．このような背景から，ソフトウェアの品質を確保しつつコスト削減を達成するために，様々な自動化技術が提案されている [?],[?],[?],[?],[?].

既存研究で提案されている EvoSuite[?] は，単体テスト自動生成における最先端のツールである．EvoSuite は，対象コードを静的解析しプログラムを記号値で表現する．そして，対象コードの制御パスを通るような条件を集め，条件を満たす具体値を生成する．単体テストを自動生成することで，開発者は手作業での作成時間が自動生成によって節約することができ，またコードカバレッジを向上することができる．しかし，既存ツールによって自動生成されるテストコードは対象のコードの作成経緯や意図に基づいて生成されていないという性質から可読性が低く開発者に信用されていないことや後の保守作業を困難にするという課題がある [?],[?],[?]．このことは、自動生成ツールの実用的な利用の価値に疑問を提示させる．テストが失敗するたびに，開発者はテスト対象のプログラム内での不具合を原因を特定するまたは，テスト自体を更新する必要があるかどうかを判断する必要がある．自動生成されたテストは，自動生成によって得られる時間の節約よりも読みづらく，保守作業に助けになるというよりかむしろ邪魔するという結果が報告されている [?].

我々は，この課題の解決するために既存テストの再利用が有効であると考える．本研究では，OSS に存在する既存の品質の高いテストコード推薦するツール SuiteRec を提案する．推薦手法の基本となるアイディアは類似コード間でのテストコード再利用である．SuiteRec は，入力コードに対して類似コードを検出し，その類似コードに対応するテストスイートを開発者に推薦する．さらに，テストコードの良くない実装を表す指標であるテストスメルを開発者に提示し，より品

質の高いテストスイートを推薦できるように推薦順位がランキングされる．

提案ツールの評価では，被験者によってSuiteRecの使用した場合とそうでない場合でテストコードの作成してもらい，テスト作成をどの程度支援できるかを定量的および定性的に評価した．その結果，SuiteRecの利用は条件分岐が多く複雑なプログラムのテストコードを作成する際にコードカバレッジの向上に効果的であること，作成したテストコードの内のテストスメルの数が少なく品質が高いことが分かった．また，実験後のアンケートによる定性的な評価では，SuiteRecを使用した場合被験者はテストコードの作成が容易になると認識し，また自分の作成したコードに自信が持てることが分かった．

2. 背景

2.1 ソフトウェアテスト

ソフトウェアテスト (以下，テスト) とは，ソフトウェア開発プロセスの中で最後の品質を確保する工程である．テストは，ソフトウェアが仕様書通りに動作することを確認すること，また不具合を検出し修正することでソフトウェアの品質を向上させることを目的として行われる．テストは図 [1] で示すように，テスト計画，テスト設計，テスト実行，テスト管理とうい大きく 4 つのタスクで構成される．テスト設計をさらに詳細に「テスト分析」,「テスト設計」,「テスト実装」のように分割する場合も存在するが，本研究ではテストケース作成に必要な作業をすべて「テスト設計」タスクとして扱う．テスト計画タスクでは，開発全体の計画に基づき，テスト対象，スケジュール，各タスクの実施体制・リソース配分等の策定を行う．テスト設計タスクでは，設計書などソフトウェアの仕様が記述されたドキュメント等を基に，テストケースを作成する．テスト実行タスクでは，ソフトウェアを動作させ，それぞれのテストケースにおいてソフトウェアが期待通りの振る舞いをするかどうかを確認する．テスト管理タスクでは，テストの消化状況やソフトウェアの品質状況の確認を随時行い，テスト優先度やリソース見直しなどのアクションを行う．テスト工程のコスト削減のため，テスト実行タスクにおいて，単体テストでは JUnit，結合テスト Selenium，Appium 等のテスト自動実行ツールの利用が進んでいる．しかし，テスト設計タスクは未だ手動で行うことが多く，自動化技術の実用化および普及が期待されている．

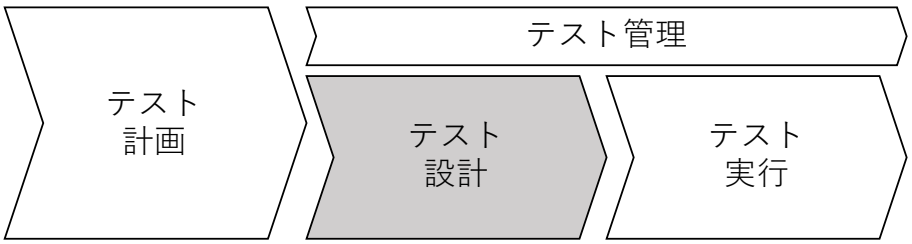


図 1 テストにおけるタスク

表 1 テストの種類

テスト粒度	説明
単体テスト	プログラムを構成する比較的小さな単位の部品が個々の機能を正しく果たしているかどうかを検証するテスト
結合テスト	個々の機能を果たすプログラムの部品 (単体) を組み合わせて，仕様通り正しく動作するかを検証するテスト
システムテスト	個々の機能や仕組みを総合した全体像のシステムとして，仕様通り正しく動作するかを検証するテスト

テスト工程は，表 1 のようにテスト対象の粒度によって単体テスト，結合テスト，システムテストの 3 種類に分類される．

単体テスト設計タスクで作成されるテストケースは，テストプロセス，テスト入力値，テスト期待値から構成される．テストプロセスに従ってテスト対象のソフトウェアにテスト入力値を与え，その出力結果をテスト期待値と比較する．これが一致していればテストは合格となり，一致しなければ不合格となる．単体テスト設計タスクにおいては多くの場合，同値分割法，境界地分析法などのテストケース作成技法を用いてテスト入力値を作成するが，ソフトウェアの要求通りに動作するかを確認するために多くのバリエーションのテスト入力値を作成する必要がある．

2.2 テストコード自動生成技術

テスト工程の支援するために様々なテストコード自動生成技術が提案されている．既存の研究 [13] は，既存のテストケースを再利用，自動生成，または再適用することによって，ソフトウェア開発のテスト工程における時間とコストを大幅に節約できることを示している．テスト生成技術は，主にランダムテスト (RT)，記号実行 (SE)，サーチベーステスト (SBST)，モデルベース (MBT)，組み合わせテストの 5 つに分類できる．SE はさらに静的記号実行 (SSE) と動的記号実行 (DSE) に分けられる．

既存研究で提案されている EvoSuite[] は，単体テスト自動生成における最先端のツールである．EvoSuite は，SBST を実装したツールであり，2011 年に発表されて以来 EvoSuite をベースとした数多くの研究がなされており，大きな影響を与えている．SBST では，一般的に以下の手順でテストスイートを生成する．

1. 達成したい要件に対する達成度合いを定量的に評価できる評価関数を設計
2. 予め用意したテストスイートをテスト対象に対して実行し，実行したテストスイートの評価関数の値を取得
3. 取得した評価関数の値が優れているテストスイートを元に，ヒューリスティック探索アルゴリズムによぅて新規にテストスイートを生成
4. 3 で生成したテストスイートをテスト対象に対して実行し，実行したテストスイートの評価関数の値を取得
5. 設定した探索打ち切り条件を満たすまで，3，4 を繰り返し実行

評価関数の設計方法は，テスト実施の観点によって異なる．例えば，SBST を用いてコードカバレッジ向上を目指したテストを実施する場合，評価関数は分岐網羅率等が用いられる．

SBST を用いたテストケース自動生成の例を提示する．図 2 において，SBST を用いて分岐 1 で「$y > 1$」を満たすようなテスト入力値を生成したい場合，評価関数をプログラムが分岐 1 に到達したタイミングで評価する．このとき，x の値が大きいほど評価関数の値も大きくなり,「$y > 1$」を満たす度合いが大きくなると定量的に評価することができる．まず，$x = -10$ として実行すると，評価関数の値は，$E = -10$ となる．続いて，仮に $x = -5$ として実行すると，評価関数の値は，$E = -5$ となる．この場合，後者のテスト入力値の方が「$y > 1$」を達成するためには優れているテスト入力値，つまり x の値が大きいテスト入力値をベースに，新しいテスト入力値の生成が行われる．それにより徐々に x の値が大きいテスト入力値が生成されていき，最終的に $x = 1$ 等のテスト入力値が取得できる．

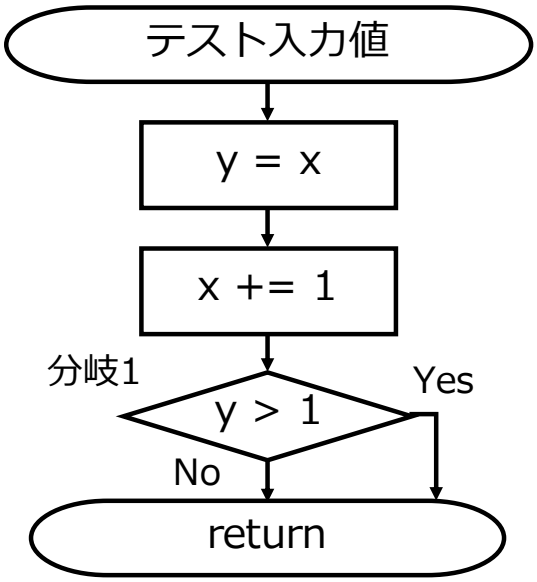


図 2 SBST によるテストケース生成の例

2.3 既存の自動生成ツールにおける課題

2.2 節では，既存の単体テスト自動生成ツール EvoSuite について説明した．単体テストを自動生成することで，開発者は手作業でのテスト作成時間を節約することができ，またコードカバレッジを大幅に向上することができる．しかし，既存ツールによって自動生成されたテストコードは，対象コードの作成経緯や意図に基づいて生成されていないという性質から可読性が低く開発者に信用されていないことや後の保守作業を困難にするという課題がある [14], [15], [16]．このことは，自動生成ツールの実用的な利用の価値に疑問を提示させる．テストが失敗するたびに，開発者はテスト対象のプログラム内で不具合の原因を特定するまたは，テスト自体を更新する必要があるかどうかを判断する必要がある．自動生成されたテストコードは，自動生成によって得られる時間の節約よりも読みづらく，保守作業に助けになるというよりかむしろ邪魔するという結果が報告されている [1].

我々は，この課題の解決するために既存テストの再利用が有効であると考える．本研究では，OSS に存在する既存の品質の高いテストコード推薦するツールを

提案する．既存テストの利用はコーディング規約や命名規則に従った可読性の高いテストコードを利用できることや，人によって作成された信頼性の高いテストコードを利用できると考える．

2.4 テストスメル

テストスメルとは，テストコードの良くない実装を示す指標である．プロダクションコードの設計だけでなく，テストコードを適切に設計することの重要性は元々Beck ら [] によって提唱された．さらに，Van Deursen ら [50] は 11 種類のテストスメルのカタログ，すなわちテストコードの良くない設計を表す実装とそれらを除去するためのリファクタリング手法を定義した．このカタログはそれ以降，18 個の新しいテストスメルを定義した Meszaros [42] によってより拡張された．

この節では，本研究で扱う以下の 6 種類のテストスメルを紹介する．

- Assertion Roulette
- Default Test
- Conditional Test Logic
- Eager Test
- Exception Handling
- Mystery Guest

以降，それぞれのテストスメルについて説明する．

2.4.1 Assertion Roulette

Assertion Roulette は，図 3 のようにテストメソッド内に複数の assert 文が存在する場合発生する．各 assert 文は異なる条件をテストするが，開発者へ各 assert 文のエラーメッセージは提供されない，そのため assert 文の 1 つが失敗した場合，失敗の原因を特定するが困難である．

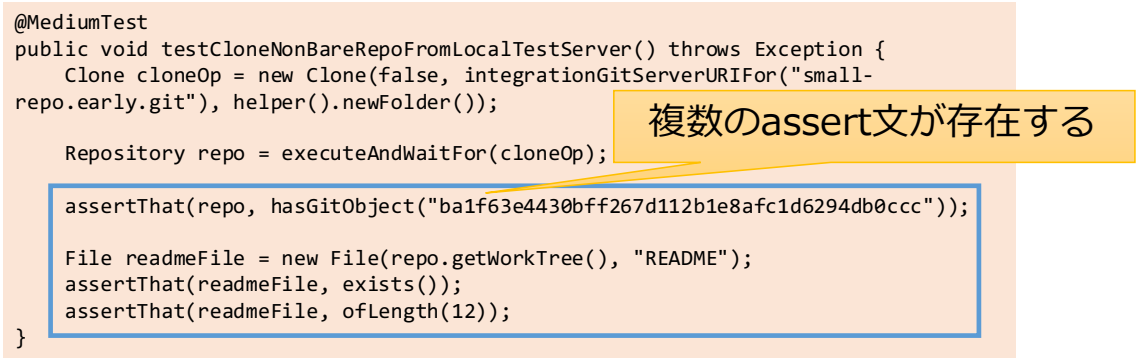


図 3　Assertion Roulette の例

2.4.2　Default Test

Default Test は，図 4 のようにテストメソッド名が初期状態 (意味のない名前) である場合発生する．テスティングフレームを使用した場合，クラス・メソッド名が初期状態である．テストコードの可読性向上のために適切な名前に変更する必要がある．

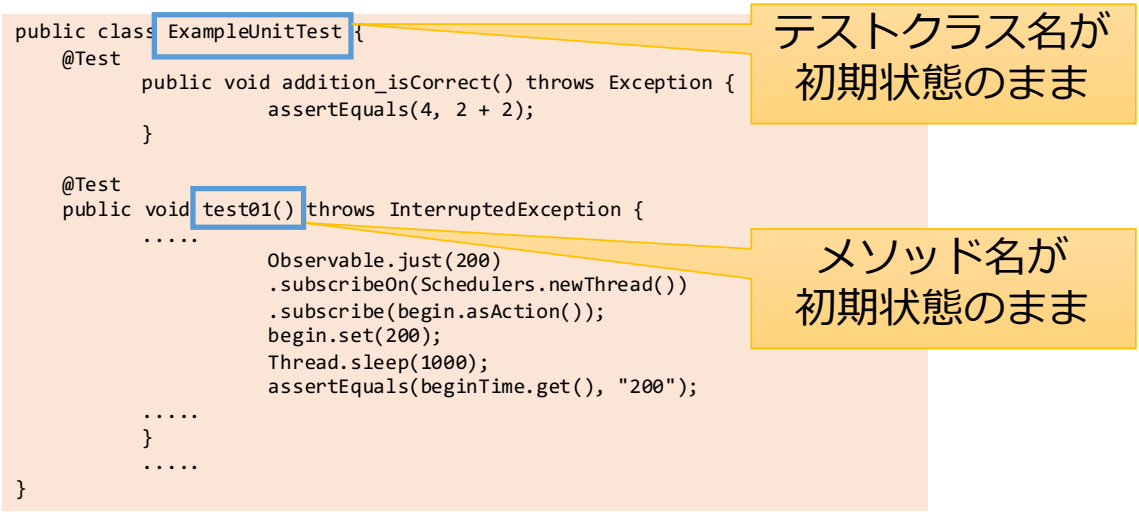


図 4　Default Test の例

2.4.3 Conditional Test Logic

Conditional Test Logic は，テストメソッド内に複数の制御文が含まれている場合発生する (図 5)．テストの成功・失敗は制御フロー内にある assert 文に基づくのでテスト結果を予測できない．また，条件分岐が多く複雑なテストコードは可読性を下げる．

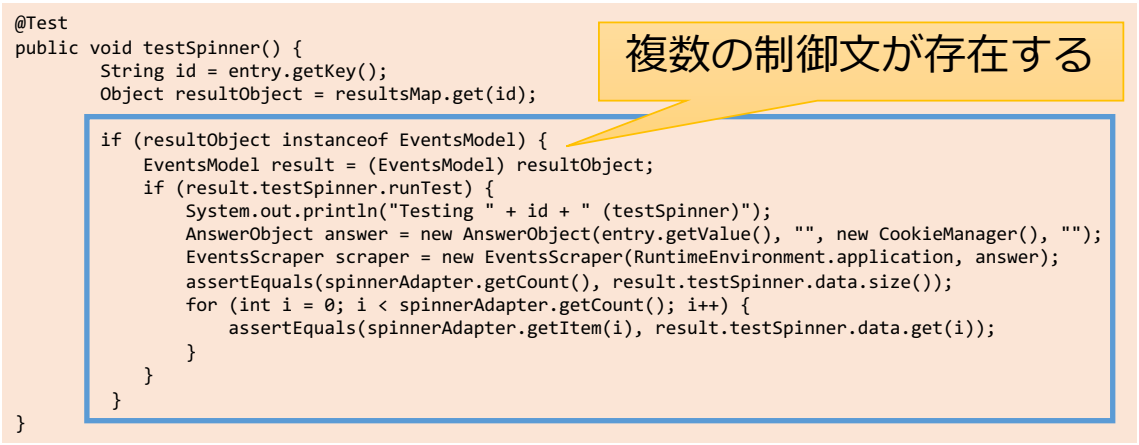


図 5　Conditional Test Logic の例

2.4.4 Eager Test

Eager Test は，テストメソッド内でテスト対象クラスのメソッドを複数回呼び出す場合発生する (図 6)．1 つのテストメソッドで複数のメソッドを呼び出すと，他の開発者はどのテスト対象をテストするのか混乱が生じる．

2.4.5 Exception Handling

Exception Handling は，テストメソッド内に例外処理が含まれている場合発生する (図 7)．例外処理は，対象コードに記述すべきで，テストコード内では正しく例外処理が行われるかを確認すべきである．

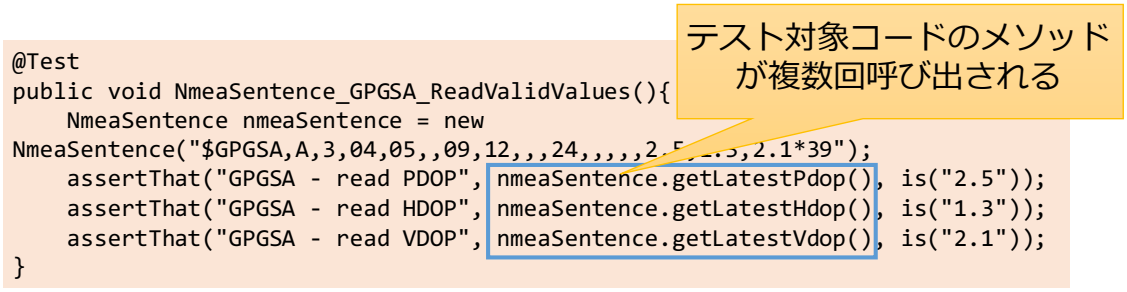
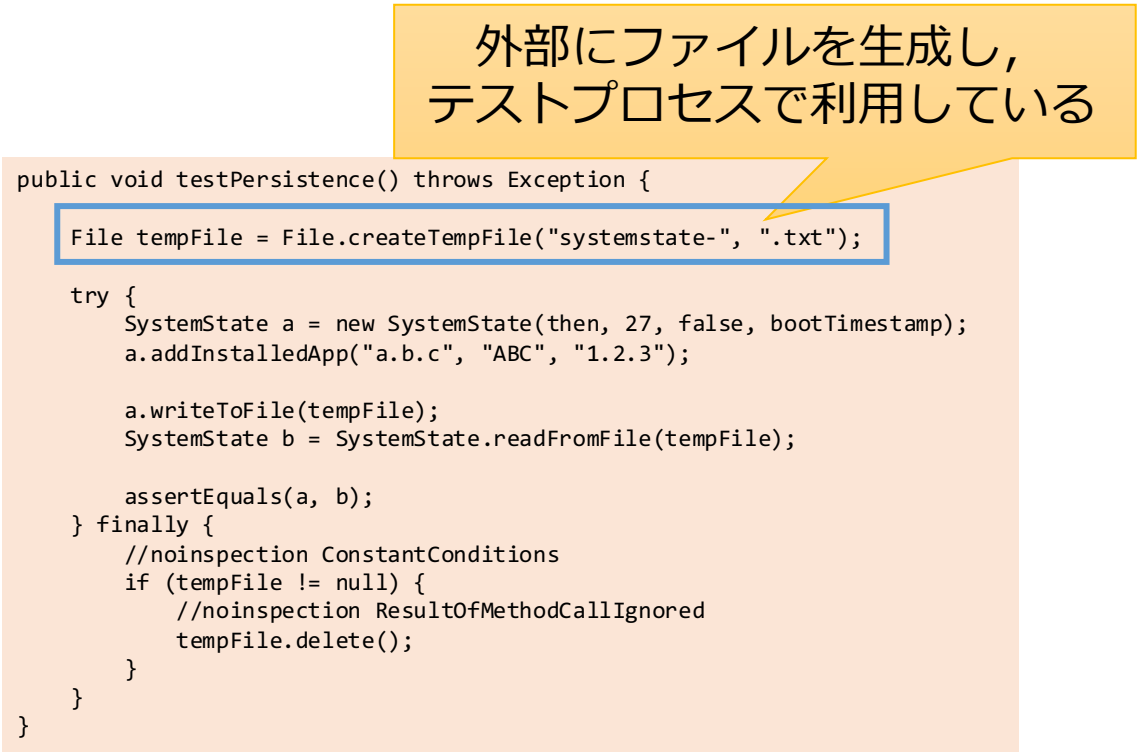


図 6 Eager Test の例

図 7 Exception Handling の例

2.4.6 Mystery Guest

Mystery Guest は，図 8 のようにテストメソッド内で外部リソースを利用した場合発生する．テストメソッド内だけでなく外部ファイルなど，外部リソースを使用すると見えない依存関係が生じる．何らかの影響で外部ファイルを削除されるとテストが失敗してしまう．




```
public void testPersistence() throws Exception {
    File tempFile = File.createTempFile("systemstate-", ".txt");

    try {
        SystemState a = new SystemState(then, 27, false, bootTimestamp);
        a.addInstalledApp("a.b.c", "ABC", "1.2.3");

        a.writeToFile(tempFile);
        SystemState b = SystemState.readFromFile(tempFile);

        assertEquals(a, b);
    } finally {
        //noinspection ConstantConditions
        if (tempFile != null) {
            //noinspection ResultOfMethodCallIgnored
            tempFile.delete();
        }
    }
}
```

図 8　Mystery Guest の例

2.5 テストスメルがソフトウェア保守にもたらす影響

2.6 コードクローン

コードクローンとは，ソースコード中に存在する同一，あるいは類似した部分を持つコード片のことであり，コピーアンドペーストなどの様々な理由により生成される．[10]．互いにコードクローンになるコード片の対のことをクローンペアと呼び，クローンペアにおいて推移関係が成り立つコードクローンの集合のことをクローンクラスと呼ぶ．これまでの研究 [1,2,8] では，コードクローンの存在はソフトウェアの保守を困難にすると言われており除去すべきと考えられていた．しかし，最近の調査ではソフトウェアの開発･保守に影響を与えるのは一部のコードクローンだけであることが明らかになり，コードクローンを除去するのではなく活用した研究も多く提案されている．

2.6.1 コードクローンの分類

既存研究 [16,24] では, クローンペア間の違いの度合いに基づき，コードクローンを以下の 4 種類に分類している．

- **タイプ 1**：空白やタブの有無，括弧の位置などのコーディングスタイル，コメントの有無などの違いを除き完全に一致するコードクローン
- **タイプ 2**：タイプ 1 のコードクローンの違い加えて，変数名や関数名などのユーザ定義名，変数 の型などが異なるコードクローン
- **タイプ 3**：タイプ 2 のコードクローンの違いに加えて，文の挿入や削除，変更などが行われたコー ドクローン
- **タイプ 4**：同一の処理を実行するが，構文上の実装が異なるコードクローン

2.6.2 コードクローン検出技術

行単位の検出 この検出手法では，ソースコードを行単位で比較し，閾値以上の長さで重複している行をコードクローンとして検出する．

字句単位の検出 字句単位の検出手法では，検出の前処理としてソースコードを字句の列に変換する．そして，閾値以上の長さで一致している字句の部分列をコードクローンとして検出する．

抽象構文木を用いた検出 抽象構文木とは，ソースコードの構文構造を木構造で表したグラフのことを意味する．この検出手法では，検出の前処理としてソースコードに対して構文解析を行うことによって，抽象構文木を構築する．そして，抽象構文木上の同形の部分木をコードクローンとして検出する．

- **行単位の検出**この検出手法では，ソースコードを行単位で比較し，閾値以上の長さで重複している行をコードクローンとして検出する．
- **タイプ 2**：タイプ 1 のコードクローンの違い加えて，変数名や関数名などのユーザ定義名，変数 の型などが異なるコードクローン
- **タイプ 3**：タイプ 2 のコードクローンの違いに加えて，文の挿入や削除，変更などが行われたコー ドクローン
- **タイプ 4**：同一の処理を実行するが，構文上の実装が異なるコードクローン