2M-8

要求定義の実現可能性保証のための シミュレーションによるテスト設計手法

式見 遼† 小形 真平‡ 松浦 佐江子†

†芝浦工業大学 理工学研究科 電気電子情報工学専攻 ‡信州大学 工学部 情報工学科

1 はじめに

システム開発プロジェクトの中断や遅延といった失敗は，主に仕様変更に伴う作業の手戻りによって発生する．手戻り発生の一因に，要求分析段階で要求定義の実現可能性を保証しないまま次の設計プロセスに進んでしまうことがある．要求定義の実現可能性を精査するには要求分析直後にシステムテストの設計を行いテスト設計の視点から要求定義の欠落や矛盾などを発見し修正する，いわゆる W モデル[1]の開発手法が有効である．W モデルでは，要求定義の問題点が発見されなくなるまでテスト設計と要求定義の修正を繰り返し行うことが望ましいが，要求定義が修正される度にテスト設計をやり直さなければならないため，作業負荷が肥大化してしまう．そのため，テスト設計が繰り返し行われなくなり，要求定義とテスト仕様の間に乖離が発生する問題がある．

そこで本稿では，要求定義の実現可能性を検証しつつ，繰り返し行なっても作業負荷が抑えられるテスト設計手法として，UML で定義した要求定義からシステムの振舞いやユーザの操作をシミュレーションすることでテストケースを生成する手法を提案する．

2 提案手法

2.1 手法概要

UML は自然言語を許容し形式表現だけで記述する言語ではないため，UML 単体をそのままシミュレーションすることは出来ない．本手法では UML に自然言語で定義された領域をソースコードに定義し直すことで UML のシミュレーションを可能にする．

提案手法によるテスト設計の流れを図 1 に示す．本手法では，顧客から獲得した要求は UML 要求分析モデルで定義する．要求分析モデルはアクティビティ図とクラス図からなり，アクティビティ図には対象システムの振舞い・ユーザの操作・システムの画面入出力の 3 つの観点でシステム利用シナリオを定義する．また，クラス図にはシステムエンティティと画面入出力で用いる画面要素のデータ構造を定義する．

要求分析モデルからツールを用いて，クラス図に定義したクラスとアクティビティ図の java スケルトンコードと，スプレッドシート形式のテストデータテンプレートを自動生成する．アクティビティ図のスケルトンコードは“シミュレーションコード”と呼び，アクティビティ図のノードに対応するメソッドと，オブジェクトノードに対応するフィールドを持つクラスである．クラス図から生成したクラスのスケルトンコードはオブジェクトノードのフィールドの型に用いられる．各ノードのメソッドにはアクティビティ図上での実行順序を指定するコードが自動的に付与され，このコードによりシミュレーション時には各ノードのメソッドがアクティビティ図上での順番に沿って逐次呼び出される．このため，ノードで行われる処理をメソッドに記述しておくことで，アクティビティ図の動作がシミュレーションできる．テストデータテンプレートはアクティビティ図ごとに生成されるスプレッドシートであり，図中の各オブジェクトノードに対応したデータを定義できる．これはテスト時にシステムで永続化されているデータやユーザの入力の想定値といったテストデータを定義するものであり，ここに定義した値はシミュレーション時に取得できる．

このように処理とデータを定義したシミュレーションコードとテストデータテンプレートを実行することで，システム内の振舞いをシミュレーションし，テストケースを生成する．

テストケースの定義項目は「テスト手順」「事前データ」「事後データ」「事前条件」「事後条件」の 5 つである．「事前データ」「事後データ」はテスト実行前後のシステム内のエンティティデータのことであり，これらはシミュレーション実行前後のデータから自動生成される．「テスト手順」はテスト時のテスターのシステム操作手順，「事前条件」「事後条件」はテスト前後にシステムが満たすべき条件のことである．後者の 3 項目は本ツールが提供する API をシミュレーションコード内で呼び出すことで記述内容を自由に設定できる．

一般的なテスト設計手法でテスト設計の繰り返しが効率的でない理由は，過去のテスト設計作業の成果を再利用出来ず毎回ゼロからやり直さなければならないからである．一方，本手法におけるシミュレーションコードとテストデータは，要求分析モデルの各要素との対応関係が明確なためモデル上で行った修正を反映させやすく再利用が容易である．そのため要求定義の変更箇所に対応する部分を修正すれば再びテストケースが生成でき，繰り返し行うテスト設計において高い効率性を発揮する．

また本手法では要求定義から自動生成したテンプレートに処理やデータを定義しているが，要求定義に不足がある場合はこれらの情報を適切に定義することができなくなる．更に，要求定義に矛盾やあいまいな記述がある場合，処理やデータを定義する際に疑問が生じる．このような兆候から要求定義の問題点が発見できる．

2.2 シミュレーションコードの生成と記述

シミュレーションコードは本手法のツールがアクティビティ図ごとに生成するコードであり，テスト設計者がアクティビティ図の各ノードで行われる処理をノードに対応するメソッドに定義することで完成する．例えば，図 2 のアクティビティ図におけるユーザの画面入力を表すアクションノード「名前を入力する」から，自動生成されるコードを図 3 に示す．図 3 の 2 行目は次に実行するメソッドを指定するコードであり，ここでは直後のジョインノードを指定している．メソッドはノードの数だ

Test Case Design Method to Guarantee the Feasibility of the Requirements Based on the Simulation.

†Ryo Shikimi ‡Shinpei Ogata †Saeko Matsuura

†Graduate School of Engineering Division of Electrical Engineering and Computer Science, Shibaura Institute of Technology

‡Department of Information Engineering, Faculty of Engineering Shinshu University

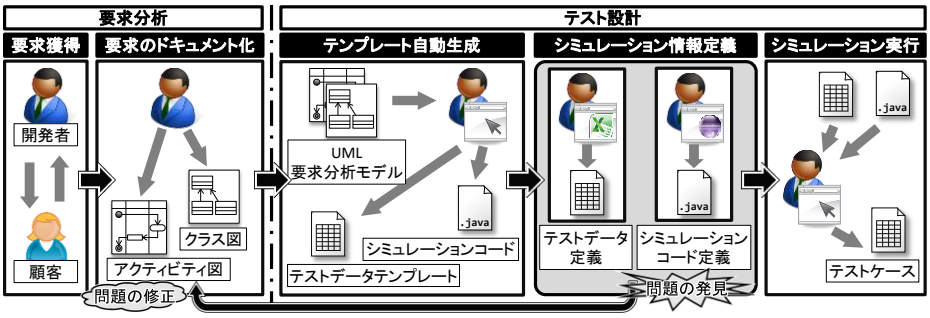

図 4 テスト設計の流れ

け生成され，テスト設計者はアクティビティ図の各ノードで行われる処理を対応するメソッドに定義する．

本手法において，テスト設計を繰り返し行うには，その都度シミュレーションコードを再生成する必要がある．この再生成では，Eclipse ASTParser を用いて既存のコードを解析し，それまでのコード情報を保ったまま要求定義の追記・削除情報のみを反映する．具体的には要求分析モデル上で追加された要素に対応するコードのみ追記し，モデル上で削除された要素に対応するコードには削除されたことを示すアノテーションを付与する．これは，再生成によってテスト設計者が定義したコードが上書きや削除されることを防ぐためである．なお，Eclipse ASTParser とは Java 開発を支援する Eclipse プラグインの集合である Eclipse Java Development Tooling[2]が提供する Java 用の抽象構文木 API であり，既存のソースコードの分析及び変更を行うことができる．

テスト設計者がメソッドに記述する処理は，「アクティビティ図内でのデータ変化」「テストデータの取得」「テストケース定義項目の設定」の 3 種類である．例えば，図 2 のアクションノード「名前を入力する」は，ユーザがオブジェクト[作成項目の入力]の属性[名前]に任意の値を代入する操作を表したノードである．すなわち，図 4 に示すようにユーザの入力の想定値をテストデータから取得し（テストデータの取得），その値でオブジェクトの値を変更する（アクティビティ図内でのデータ変化），ユーザの操作手順の 1 つ（テストケース定義項目の設定）であるため，このノードに対応するメソッドには 3 種類の処理を全て定義する．

2.3 データ変化の手動定義

手続きの複雑な要求に対して，シミュレーションコードを定義することは非常に難しい．例えば，開発システムで自然言語処理を用いる場合などがこれに当たる．このような処理は記述すると複雑だが，処理の満たすべき性質は明らかであるため，テスト設計者が処理結果を直接考えるほうが，作業効率が良い．

そこで，本手法では処理をコーディングする代わりに，実行時に処理で行われるデータの変化を手動で定義することを可能にした．具体的には，シミュレーションコード定義の段階でコーディングが困難な処理部分にデータ変化を手動で定義するためのメソッドを呼び出すコードを記述する．シミュレーション実行時にこのコードが実行されると，データ編集ダイアログが表示され，ダイアログ上で処理後のデータを直接入力することができる．

3 まとめ

本手法のテンプレートの自動生成等を行う支援ツールは Eclipse プラグインとして実装した．これは，手法のテスト設計作業を全て Eclipse で行えるようにして，作業効率を高めるためである．

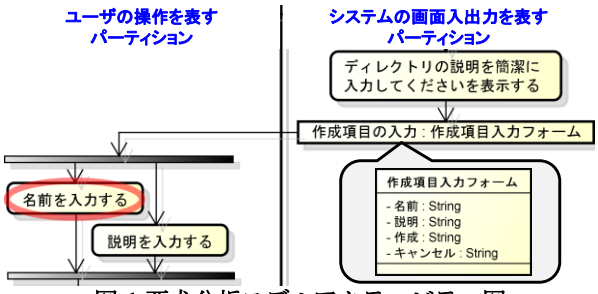

図 1 要求分析モデルアクティビティ図

```
1  public void actNode$名前を入力する() {
2    simulator.setNext("joinNode$合流 0");
3  }
```

図 2 自動生成されたシミュレーションコード

```
1  public void actNode$名前を入力する() {
2    // テストデータからユーザの入力値を取得
3    作成項目入力フォーム userInput
4          = testdata.get_作成項目の入力();
5    // [作成項目の入力]オブジェクトに入力値を与える
6    this.作成項目の入力.名前 = userInput.名前;
7    // テストケースにテスト手順を追加
8    simulator.addTestcaseStep
9      ("ユーザは名前に" + userInput.名前 + "を入力する");
10   simulator.setNext("joinNode$合流 0");
11 }
```

図 3 シミュレーションコード定義例

本稿で提案したシミュレーションによるテスト設計手法を，現在大学で運用されている web 授業支援システムの機能拡張案件に適用した．その結果，本手法により目視では発見できなかった要求定義の不足，矛盾，あいまいな表現の問題が多数発見でき，支援ツールによるテスト設計の効率化も確認できた．

UML からテストケースの生成を行う研究[3][4]が多数あるが，いずれもテスト設計のフィードバックによる要求定義の改善を考慮していない．本稿では要求定義の実現可能性精査を目的としたシミュレーションによるテスト設計手法と手法を支援するツールを示した．課題は，2.2 章の再生成をシミュレーションコードだけでなくテストデータテンプレートにも適用させることである．

― 参考文献 ―

[1] A. Spillner. The W-MODEL. Strengthening the Bond Between Development and Test. In Int. Conf. on Software Testing, Analysis and Review, 2002.

[2] Eclipse, Eclipse Java development tools (JDT), http://www.eclipse.org/jdt/.

[3] L. Wang, J. Yuan, X. Yu, J. Hu, X. Li, and G. Zheng：Generating Test Cases from UML Activity Diagram based on Gray-Box Method.

[4] S. Weißleder and B.-H. Schlingloff. Deriving Input Partitions from UML Models for Automatic Test Generation. In LNCS Volume on Models in Software Engineering, 2007.