研究室紹介

ソフトウェアのテストについて

廣田 豊彦
Toyohiko HIROTA
九州産業大学 情報科学部 社会情報システム学科
Faculty of Information Science, Kyushu Sangyo University
hirota@is.kyusan-u.ac.jp, http://www.is.kyusan-u.ac.jp/~hirota/

1. ソフトウェア工学

現在では，ソフトウェアという語を耳にしたことがない人は少ないでしょう．一般の家庭でも PC があり，インターネットに接続されているというのが珍しくない時代です．ソフトウェアはハードウェアと対になる語と考えられていますが，コンピュータ以前の時代にはハードウェアという語しかありませんでした．それは金物を意味していました．コンピュータが登場して，その物理的実体がハードウェアと呼ばれるようになり，それと対比する形でソフトウェアという新しい語が作られたようです．

ソフトウェアが一般化していることと比較すると，ソフトウェア工学（あるいはソフトウェアエンジニアリング）はあまり知られていないように思われます．ソフトウェア工学とは，ソフトウェアの開発，運用，保守への工学的アプローチとそれに関する研究を指しています [1]．工学は，単なる技術の集まりではなく，背景となる理論体系や，組織的な教育方法が確立していなければなりません．

ソフトウェアの開発は大きく 3 つのフェーズに分けて考えることができます．要求分析，設計，そして実装です [2]．ソフトウェアはきわめて多様な能力を持つことができます．そこで，現実の資源，すなわち開発経費やハードウェアの能力に見合う適切なソフトウェアを開発する必要があります．そのためには顧客の要求をきちんと整理しなければなりません．それが要求分析のフェーズで，その結果として要求仕様書が作成されます．顧客が期待するソフトウェアと，開発者が約束するソフトウェアの合意が要求仕様書です．

要求仕様書に基づいてソフトウェアを開発するわけですが，すぐにプログラムを書き始めるわけにはいきません．一般にソフトウェア開発には多数の技術者が関わります．そこで，それらの人々の間で，開発のための基本方針を定めておかなければなりません．それが設計のフェーズで，その結果として設計仕様書が作成されます．実際には，設計は単一のフェーズではなく，開発するソフトウェア全体の設計から，ある一部分の詳細な設計まで多くのレベルに分割されます．

一般に製品は多数の部品からできあがっています．それらの部品もさらにより細かな部品に分解できることが多いでしょう．ソフトウェアの場合でも，そのように分割して開発します．設計フェーズにおいて，ソフトウェアをどのように分割するかを決定します．このとき分割された部品同士をどのように結合するかをはっきりさせておくことも重要です．

設計書ができると，それに従って実装，すなわちプログラミングを行います．初心者にはプログラミングはたいへん困難な作業のように思えますが，ソフトウェア開発全体から見ると，ほんの一部の作業にすぎません．しかも過去に多くのプログラムが書かれており，それらを有効に利用することで，新たに書かなければならないプログラムの量をを減らすことができます．むしろ，そういった過去の資産を活用できるように，いかにうまく設計するかが重要になっています．

以上，ソフトウェア開発の 3 つのフェーズについて簡単に紹介しましたが，これらの 3 つのフェーズが順に実行されるわけではありません．かつては，これらのフェーズが順に，後戻りすることなく実行されることが理想とされていました．しかし現在では，そのような理想的な開発スタイルは非現実的であることが認識されています．実際には，これらのフェーズがオーバーラップしたり，3 つのフェーズを何度も反復したり，といったやり方でソフトウェア開発が行われます．

ソフトウェアは多種多様な能力を持つことができます．その裏返しとして，ソフトウェアには多種多様な欠陥がありえます．ここで欠陥とは，ソフトウェアが顧客の期待通りに動作しないことを指します．要求仕様書通りに開発したはずなのに，実際にはその通りに動かないとすると，それは欠陥です．実は要求仕様書自体が欠陥を含んでいることも珍しくありません．

要求仕様書は，顧客と開発者が合意したものであると最初に説明しましたが，顧客はソフトウェア開発の専門家ではないため，要求仕様書に記述されている内容を十分に理解できるとはかぎりません．一方，開発者は顧客のやりたい仕事の内容を熟知しているわけではありません．そのため，顧客が考えていることを的確に要求仕様書に表現することは容易ではありません．

ソフトウェアの欠陥を発見し修正するためのアプロー

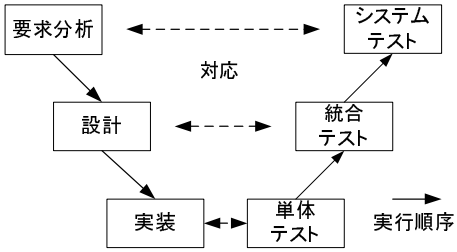

図 1 ソフトウェア開発フェーズとテスト

チは，大きく分けて二つあります．一つは，開発の途中で，欠陥の原因が入り込むことがないようにするというアプローチです．もう一つはできあがった製品をテストし，欠陥を発見するというアプローチです．両者はどちらか一方でよいというものではありません．欠陥の原因を作らないようにソフトウェアを開発するための方法論もいろいろと提案されてきましたが，だれでもできるわけではない，どんなソフトウェア開発へも適用できるわけではない，といった課題があります．

一方，テストに関しては，ソフトウェアの能力をすべてテストすることは事実上不可能であるという根本的な課題があります．テストをより多く実施することでソフトウェアの信頼性を高めることができますが，そのためのコストや期間が増大します．いかに効率的かつ効果的にテストを行うかは，ソフトウェア工学の重要な課題の一つです．

2. ソフトウェアのテスト

テストと一口に言っても，様々なものがあります．テストの分類方法自体，一通りではありません．ソフトウェア開発フェーズに対応するテストの分類として，単体テスト，統合テスト，システムテストがあります．単体テストは，実装の直後に，実装の基本単位毎に行うテストです．統合テストは，設計フェーズでのソフトウェアの分割を逆にたどって，順にソフトウェア部品を組み立てながらテストを行います．システムテストは，ソフトウェア全体をテストするもので，ソフトウェアが要求仕様書通りに動作するかどうかを確認します．

ソフトウェア開発フェーズとテストの対応関係を図 1 に示します．ここで注目すべきことは，開発フェーズの時間的順序とテストの時間的順序が逆になるということです．要求仕様書に欠陥の原因がある場合，その欠陥がテストによって発見されるのは，最後のシステムテストのときです．そして，そのような欠陥が発見されると，開発フェーズの最初である要求分析からやり直すことが必要になります．このような欠陥と比較すると，実装フェーズのプログラミングに起因する欠陥の除去ははるかに容易なものと言えます．

ソフトウェアテストの分類方法の一つとして，ブラックボックステストとホワイトボックステストがあります．これはソフトウェアのソースコードを参照するかどうかの違いによるものです．ソースコードを参照せずに，ソフトウェアの仕様書（要求仕様書や設計仕様書）だけを参照して，テストを計画し，実行するやり方を，ブラックボックステストと呼びます．中身の見えない箱のことをブラックボックスと呼ぶことから名づけられたものです．一方，ソースコードを参照するテストのやり方をホワイトボックステストと呼びます．ホワイトボックスという語は一般的な言葉ではありませんが，ブラックボックスとは逆の意味ということで名づけられたものです．

本来のテストの目的，すなわち欠陥を発見するという立場からは，ブラックボックステストだけで十分なはずです．仕様書通りにソフトウェアが動作するかどうかを発見するのがテストの目的だからです．それにもかかわらずホワイトボックステストという考え方があり，実際にそのようなテストが行われます．それはソフトウェアのテストの困難さに原因があります．ソフトウェアが仕様書通りに動作するかどうか，完全にテストすることは事実上不可能です．そこで効果的なテストを選択して実施する必要があります．そのときにソースコードを参照することで，効果的なテストを選択する，あるいはテストの効果を測定することができます．

ところで，ブラックボックステストとホワイトボックステストという分類はあまり適切な分類ではありません．単体テスト，統合テスト，システムテストのうちで，この分類が意味を持つのは単体テストの場合だけです．統合テストやシステムテストをソースコードに基づいて実施することは事実上不可能であり，これらのテストは必然的にすべてブラックボックステストになるからです．また，ソースコードを参照するといっても，実はいろいろなアプローチがあります．そこで，テストモデルという考え方を導入します [3]．テストモデルとは，テスト対象のソフトウェア（あるいはその一部）を，テストの視点からモデル化したものです．テストモデルを導入することで，何をテストしようとするのかを明らかにすることができます．

単体テストのためのテストモデルとして，制御フローモデル，データフローモデル，状態遷移モデルなどがあります．オブジェクト指向プログラムがテスト対象となる場合，制御フローモデルとデータフローモデルは一つのメソッドをテストします．そして状態遷移モデルはクラス全体をテストします．

制御フローモデルは，ソースコードのどの部分がどのような順序で実行されるかをモデル化したものです．制御フローモデルは，ノードとエッジからなるフローグラフで表されます．ノードには分岐と合流の 2 種類があり，他に開始ノードと終了ノードが一つずつあります．図 2 のメソッド numOfSubs の制御フローモデルを図 3 に示します．制御フローモデルでは，フローグラフ中のすべ

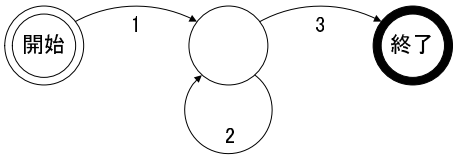

図 3 制御フローモデル

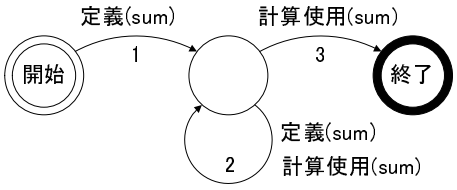

図 4 データフローモデル

てのエッジをテストする，というのが一般的なテストの規準です．図 3 でその規準を満たすもっとも簡単なテストは，ループを 1 回だけ実行して終了するような経路を実行するものです．

制御フローモデルでは，すべてのノードは，開始ノードから終了ノードの経路上に存在しなければなりません．あたりまえのようですが，現実には孤立したノードが存在することがあります．それは決して実行されることのないソースコードが存在することを意味しています．プログラムの修正を繰り返すうちに，以前は使われていたコードが使われなくなったというようなことがよくあります．しかし，ほんとうに不必要なのかどうか，十分に検討する必要があります．もしもそのコードが実は必要なコードであったとすると，それはソフトウェアの欠陥につながることになります．このように，テスト実行以前のモデル作成段階で，欠陥の原因が見つかることもあります．

データフローモデルは，制御フローモデルで用いたフローグラフのエッジに，次の 4 種類の注釈を付加したものです．

- データの定義
- データの消滅
- データの計算使用
- データの述語使用

データの定義とは，変数にデータが代入されることを意味します．データの消滅は，内側のブロックで宣言された変数が，ブロック外で無効になるような場合です．データの計算使用とは，ある変数が別の変数（あるいは同じ変数）の値を計算するために使われることを意味します．データの述語使用とは，ある変数が分岐条件を判定するために使われることを意味します．図 4 にデータフローモデルの例を示します．

データフローモデルでのテストとしてもっとも典型的なものは，可能なすべての定義と使用の組み合わせをテストするというものです．たとえば，図 4 の変数 sum に着目すると，エッジ 1 と 2 に定義があり，エッジ 2 と 3 に使用があります．そこですべての組み合わせは

(a) エッジ 1 の定義とエッジ 2 の使用
(b) エッジ 1 の定義とエッジ 3 の使用
(c) エッジ 2 の定義とエッジ 2 の使用
(d) エッジ 2 の定義とエッジ 3 の使用

の 4 通りになります．これらをすべて実行するためには，少なくとも次の 3 回のテストを実行します．

(1) ループを 1 度も実行しない．(b)
(2) ループを 1 度だけ実行する．(a，d)
(3) ループを 2 度以上実行する．(c)

制御フローモデルでは，2 だけでよかったのですが，データフローモデルを採用すると，1 や 3 もテストする必要があります．

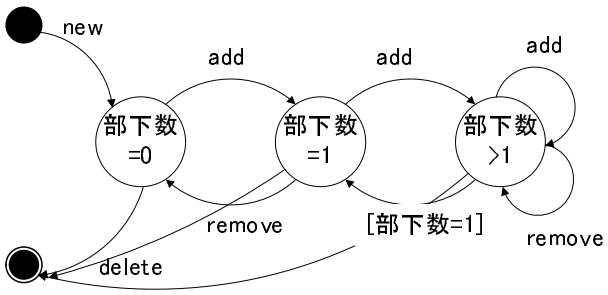

図 5 状態遷移モデル

データフローモデルでも，テスト実行前に欠陥の原因が見つかることがあります．データが定義されていないのに使用される，といった場合です．一般にこのような誤りをデータフロー異常といいます．特に，ある特定の条件のときだけ発生するデータフロー異常に注意する必要があります．

データフローモデルは単一のメソッドを対象としますが，変数がクラス属性の場合，メソッド間での定義と使用の関係が問題となります．これについては，ここでは立ち入らないことにします [4].

状態遷移モデルは，オブジェクトの内部状態に関する状態遷移図です．状態遷移は，メソッドの呼び出しや，内部状態に関する条件の成立によって起こります．外部からのイベントによっても状態遷移が起こりえますが，一般のオブジェクト指向プログラムでは，メソッド呼び出しと同じになります．図 2 のクラス Employee の状態遷移モデルを図 5 に示します．

状態遷移モデルにおけるテストの規準は，すべての状態遷移を 1 度以上起こす，というものです．図 5 の例では，

(1) create add delete
(2) create add remove delete
(3) create add add add delete
(4) create add add remove delete

の 4 通りになります．

状態遷移モデルでは，原則としてすべての状態と状態

```
import java.util.*;

public class Employee {
  Vector employees;

  public boolean add(Employee e) {
    employees.add(e);
    return true;
  }
  public void remove(Employee e) {
    employees.removeElement(e);
  }
  public int numOfSubs() {
    int sum = 0;
    for (int i = 0; i < employees.size(); i++) {
      sum += ((Employee)employees.elementAt(i)).numOfSubs() + 1;
    }
    return sum;
  }
}
```

図 2 例題クラス

遷移の組み合わせが網羅されている必要があります．オブジェクトが一旦生成されたら，その後は任意のメソッドが呼び出される可能性があります．またそのオブジェクトがどのような状態であっても，破棄される可能性があります．状態遷移図ですべての可能性が網羅されていない場合，状態遷移モデルの作り方が誤っていたか，あるいは仕様書に誤りがあったことになります．いずれにしてもテストを実行する以前にそれらをもう一度検討する必要があります．

3. 廣田研究室の研究テーマ

廣田研究室の基本テーマはソフトウェア工学です．前節で紹介したソフトウェアテストに関しては，単体テストを自動化するための技術を開発しました [5]．最近は，要求分析の一環として，ビジネスプロセスモデリング [6] や，データベーススキーマ設計 [7] などの研究を行っています．また，対象領域の特質を把握してソフトウェア開発を効率化するために，ドメイン分析・モデリングの研究を行ってきました [8, 9, 10]．具体例として，建築図面の自動認識や，建築プロセスの分析と支援ツールの開発などがあります．近年は分析・モデリングといった抽象的な研究が中心になってきましたが，Java，C++，Prolog，Tcl/Tk，Python，Visual Basic など，様々なプログラミング言語を駆使して，便利なツールを開発することにも取り組んで行きたいと考えています．

◇ 参 考 文 献 ◇

[1] SWECC. *Guide to the Software Engineering Body of Knowledge, Version 0.95*. IEEE, 2001. (邦訳：『ソフトウェアエンジニアリング基礎知識体系－ SWEBOK －』松本吉弘監訳，オーム社，2003 年).

[2] Shari Lawrence Pfleeger. *Software engineering: Theory and Practice*. Prentice-Hall, 1998. (邦訳：『ソフトウェア工学：理論と実践』 堀内泰輔訳，ピアソン・エデュケーション，2001 年).

[3] Shel Siegel. *Object Oriented Software Testing*. John Wiley & Sons, 1996. (邦訳：『オブジェクト指向ソフトウェアテスト技法』 古宮， 廣田監訳，共立出版，1998 年).

[4] 廣田 豊彦, 橋本 正明. オブジェクト指向プログラムのためのデータフローテスト. **電子情報通信学会論文誌** *D-I*, J97-D-I(10):707–718, 1996 年 10 月.

[5] 山口 嘉文, 伊藤 正之, 足立 浩二, 廣田 豊彦, 橋本 正明. 階層的漸増テスト自動実行ツール. In **ソフトウェア工学の基礎** *VII*, pages 181–188. 近代科学社, 2000.

[6] 廣田 豊彦, 熊谷 敏, 川端 亮, 伊藤 潔. ダイアグラム変換を利用したシステム分析. **電子情報通信学会技術研究報告（知能ソフトウェア工学）**, 104(49):7–12, 2004 年 5 月.

[7] 廣田 豊彦, 熊谷 敏, 稲永 健太郎. ユースケースに基づくデータベーススキーマ設計法. **電子情報通信学会技術研究報告（知能ソフトウェア工学）**, 102(503):19–23, 2002 年 12 月.

[8] 伊藤 潔, 杵嶋 修三, 田村 恭久, 廣田 豊彦, 吉田 裕之 編. **ドメイン分析・モデリング**. 共立出版, 1996.

[9] K. Itoh, T. Hirota, S. Kumagai, and H. Yoshida, editors. *Domain Oriented Sytems Development: Principles and Approaches*, volume 1 of *Advanced Information Processing Technology*. Gordon and Breach, 1998.

[10] K. Itoh, S. Kumagai, and T. Hirota, editors. *Domain Oriented Sytems Development: Perspective and Practices*, volume 6 of *Advanced Information Processing Technology*. Taylor & Francis, 2002.